US006356559B1

(12) **United States Patent**
Doucette et al.

(10) **Patent No.: US 6,356,559 B1**
(45) **Date of Patent: *Mar. 12, 2002**

(54) **COMBINED SYNCHRONOUS AND ASYNCHRONOUS MESSAGE TRANSMISSION**

(75) Inventors: **John Doucette**, Londonderry; **Thomas J. Bryden**, Peterborough; **Todd Byron**, Manchester, all of NH (US)

(73) Assignee: **At Comm Corporation**, San Mateo, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/578,554**

(22) Filed: **May 25, 2000**

Related U.S. Application Data

(63) Continuation of application No. 09/268,099, filed on Mar. 13, 1999.
(60) Provisional application No. 60/098,297, filed on Aug. 27, 1998.

(51) **Int. Cl.**[7] **H04L 12/42**; H04L 12/403; H04L 12/66; H04J 3/16; H04J 3/22
(52) **U.S. Cl.** **370/450**; 370/353; 370/468
(58) **Field of Search** 370/352–354, 370/449, 450, 452, 460, 468, 470–473, 476, 909; 709/238, 251

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,068,849	A *	11/1991	Tamaka	370/509
5,245,605	A *	9/1993	Ofek	370/447
5,570,355	A *	10/1996	Dial et al.	370/352
5,935,214	A *	8/1999	Stiegler et al.	709/238
6,108,346	A *	8/2000	Doucette et al.	370/450
6,233,251	B1 *	5/2001	Kurobe et al.	370/741

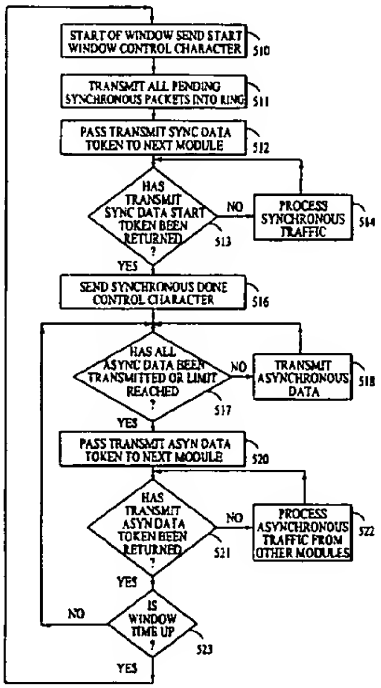
* cited by examiner

Primary Examiner—Huy D. Vu
Assistant Examiner—Kevin C. Harper
(74) *Attorney, Agent, or Firm*—Elmer Galbi

(57) **ABSTRACT**

A communication system including a collection of modules coupled in a ring architecture which integrates synchronous and asynchronous message transmission. Asynchronous data packets and synchronous voice packets are exchange on a single communication link. Packetized information exchange references a fixed length window with all synchronous data being exchanged at the beginning of each window and with asynchronous data exchanged during the remaining portion of each window. Virtual circuits utilizing the synchronous packets can deliver telephone conversations without degradation in voice quality and yet the system can also transmit asynchronous data packets.

14 Claims, 5 Drawing Sheets



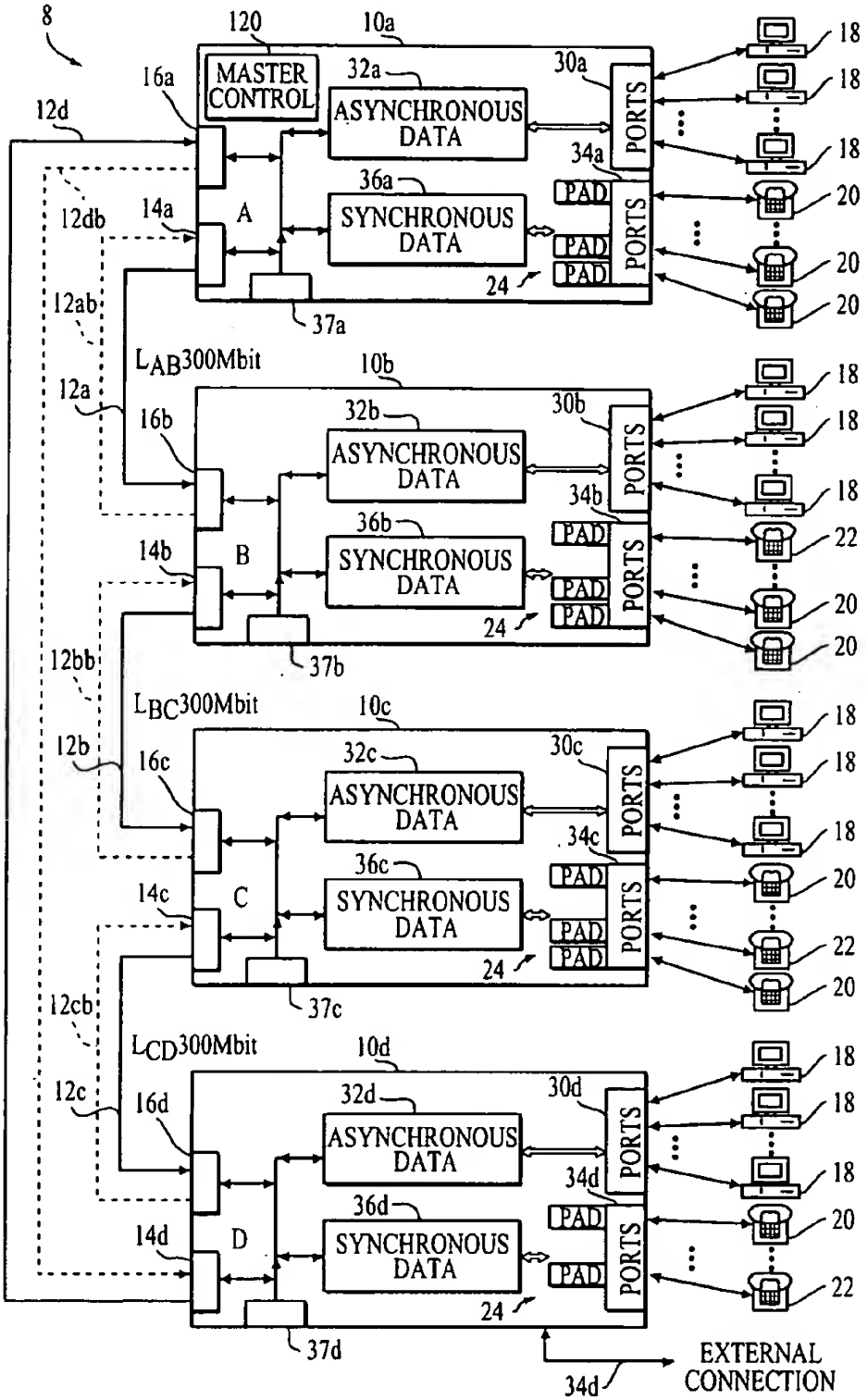

8
120
10a
MASTER CONTROL
16a
12d
32a
ASYNCHRONOUS DATA
30a
PORTS
34a
36a
SYNCHRONOUS DATA
PAD
12db
14a
A
24
37a
12ab
L_{AB}300Mbit
10b
12a
16b
32b
30b
34b
36b
14b
B
37b
12bb
L_{BC}300Mbit
10c
12b
16c
32c
30c
34c
36c
14c
C
37c
12cb
L_{CD}300Mbit
10d
12c
16d
32d
30d
34d
36d
14d
D
37d
18
20
22
EXTERNAL CONNECTION

FIG. 1

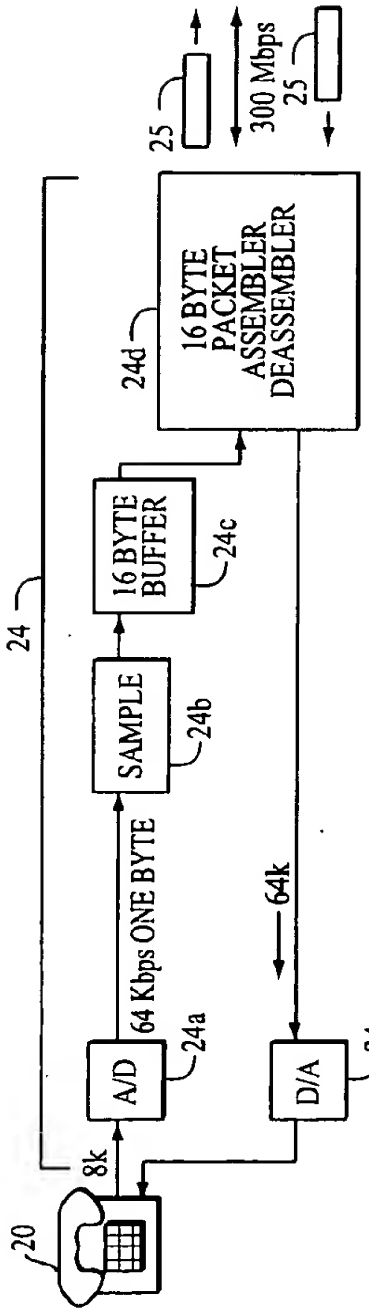

20
8k
A/D
24a
64 Kbps ONE BYTE
SAMPLE
24b
16 BYTE BUFFER
24c
64k
D/A
24e
24
24d
16 BYTE PACKET ASSEMBLER DEASSEMBLER
25
300 Mbps
25

FIG. 2

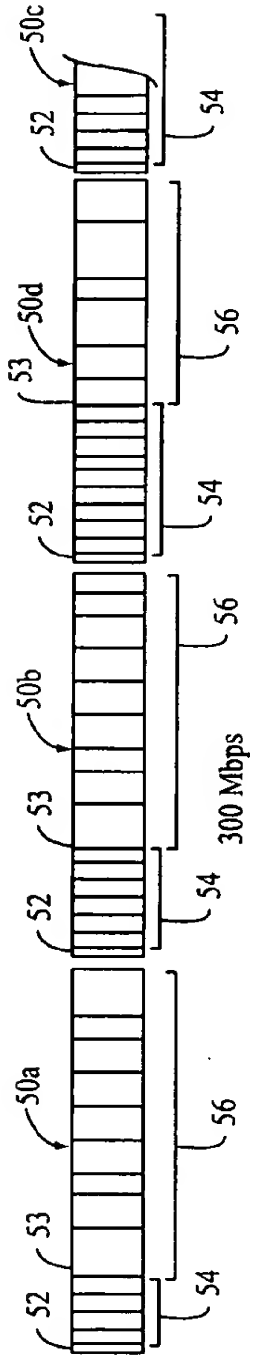

50a
50b
50c
50d
52
53
54
56
300 Mbps

FIG. 3

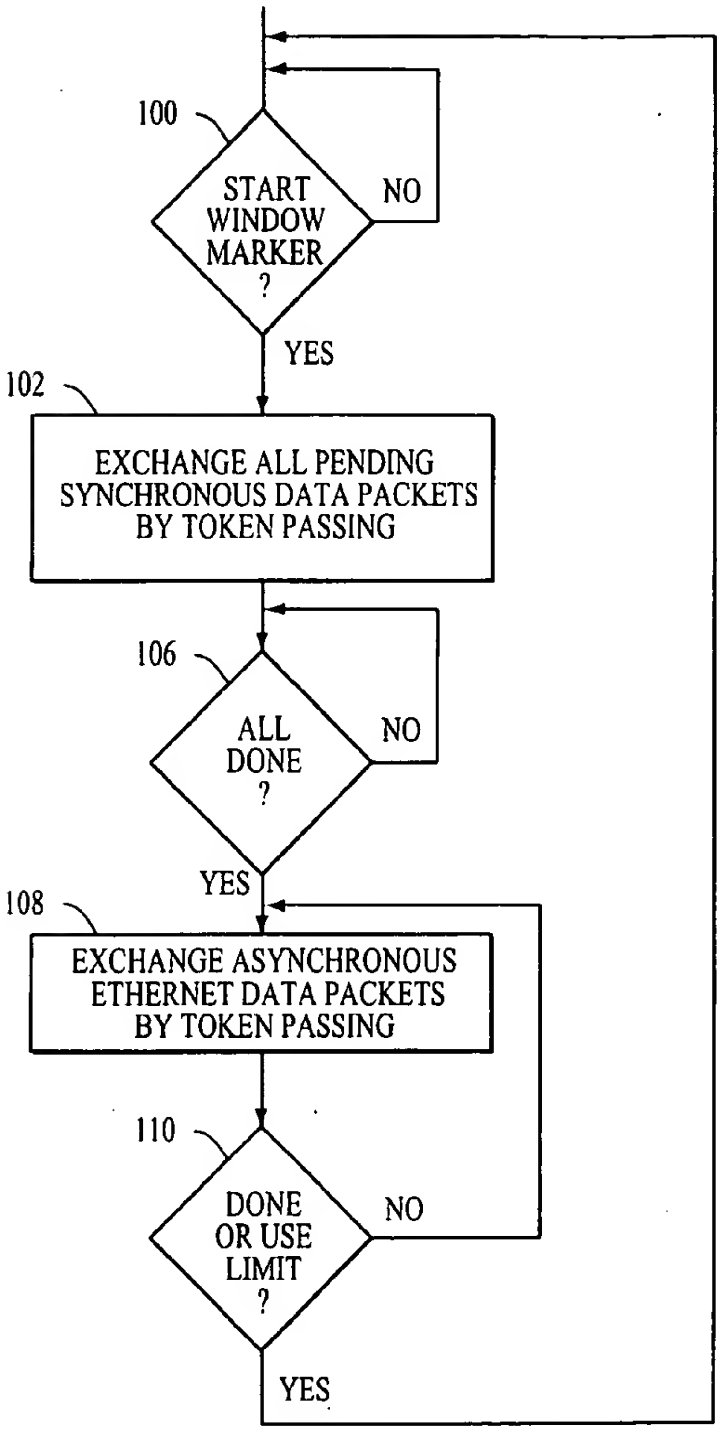

100
START WINDOW MARKER ?
NO
YES
102
EXCHANGE ALL PENDING SYNCHRONOUS DATA PACKETS BY TOKEN PASSING
106
ALL DONE ?
NO
YES
108
EXCHANGE ASYNCHRONOUS ETHERNET DATA PACKETS BY TOKEN PASSING
110
DONE OR USE LIMIT ?
NO
YES

FIG. 4

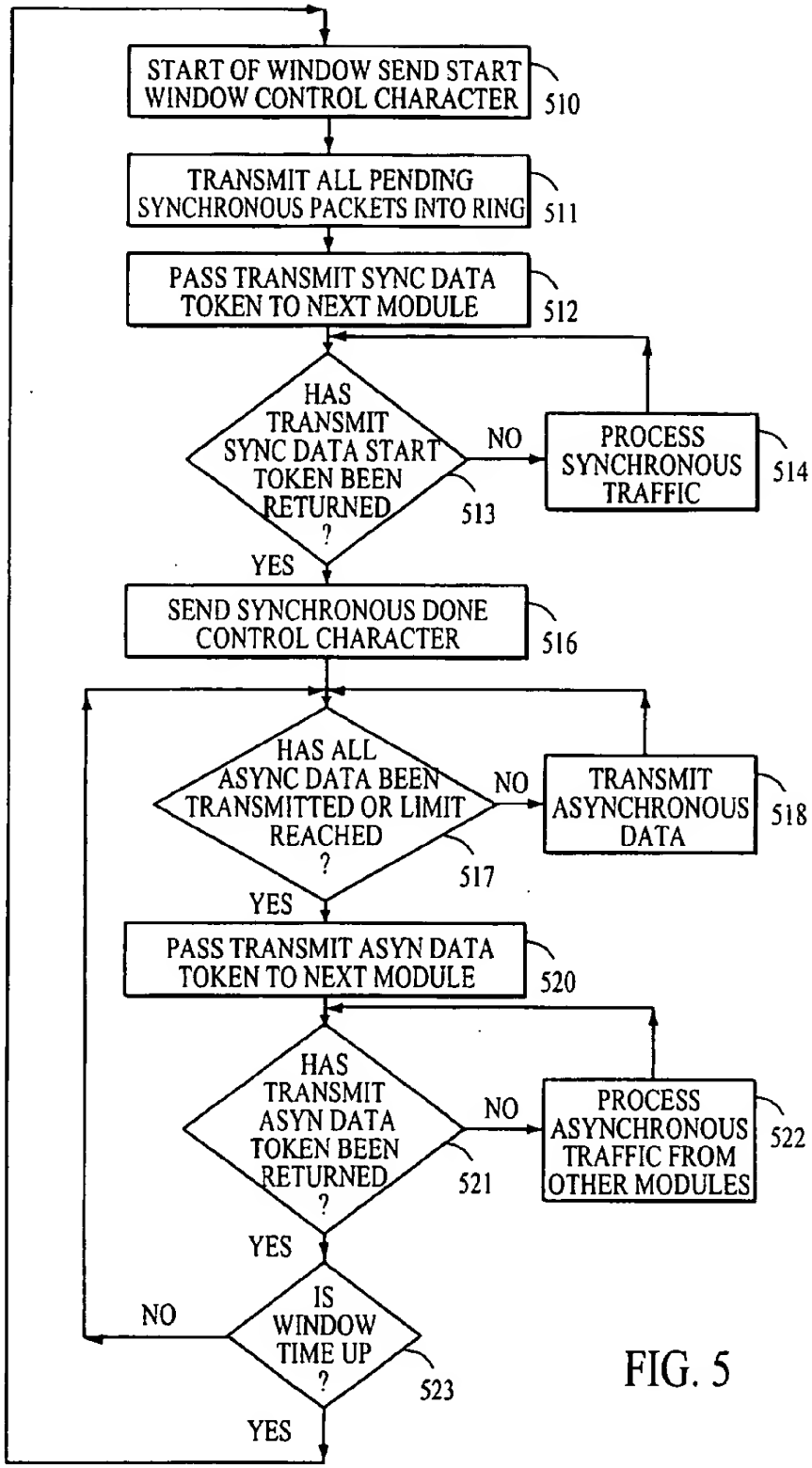
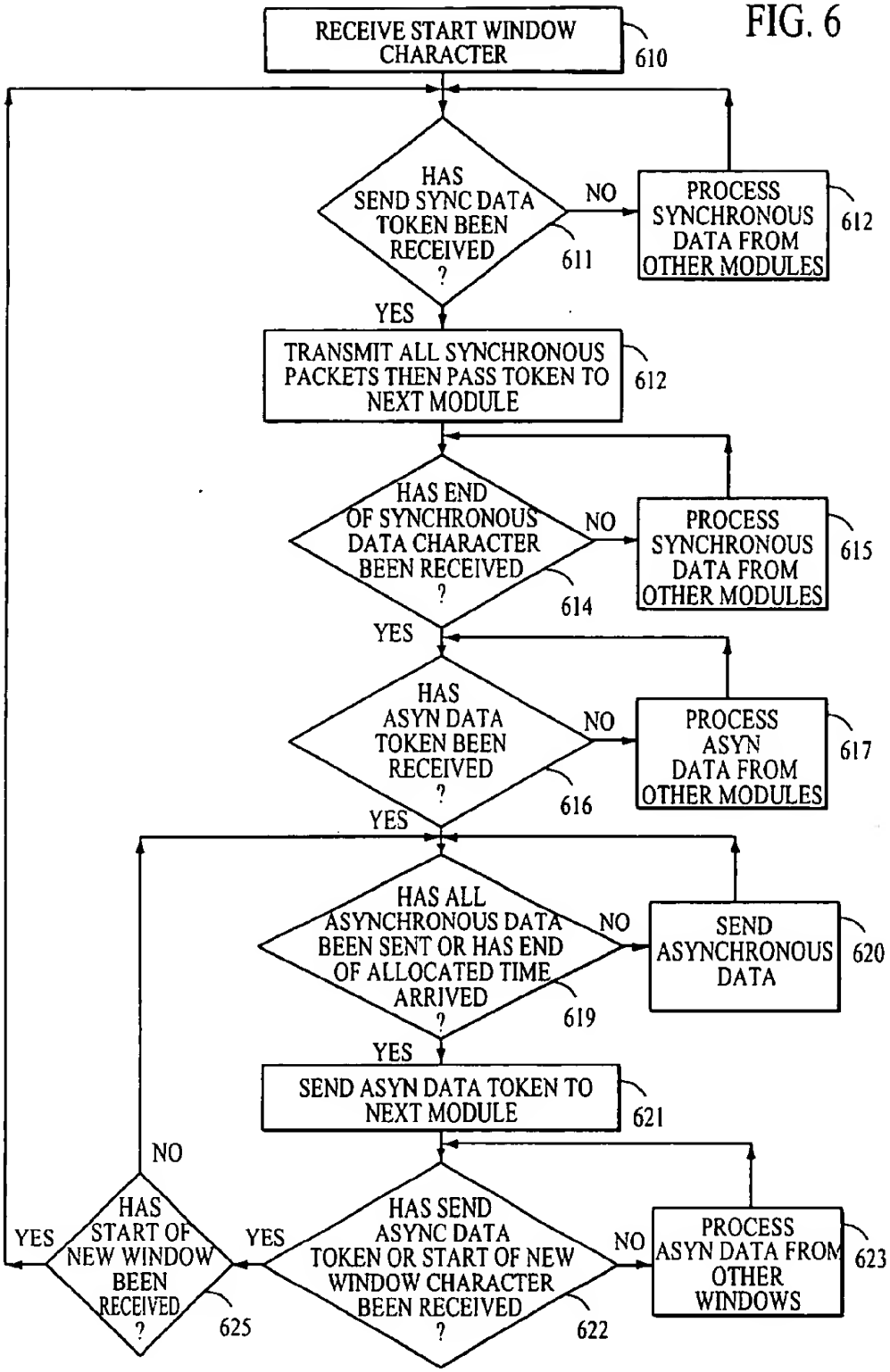

START OF WINDOW SEND START WINDOW CONTROL CHARACTER
510
TRANSMIT ALL PENDING SYNCHRONOUS PACKETS INTO RING
511
PASS TRANSMIT SYNC DATA TOKEN TO NEXT MODULE
512
HAS TRANSMIT SYNC DATA START TOKEN BEEN RETURNED ?
513
NO
PROCESS SYNCHRONOUS TRAFFIC
514
YES
SEND SYNCHRONOUS DONE CONTROL CHARACTER
516
HAS ALL ASYNC DATA BEEN TRANSMITTED OR LIMIT REACHED ?
517
NO
TRANSMIT ASYNCHRONOUS DATA
518
YES
PASS TRANSMIT ASYN DATA TOKEN TO NEXT MODULE
520
HAS TRANSMIT ASYN DATA TOKEN BEEN RETURNED ?
521
NO
PROCESS ASYNCHRONOUS TRAFFIC FROM OTHER MODULES
522
YES
IS WINDOW TIME UP ?
523
NO
YES

FIG. 5

FIG. 6


RECEIVE START WINDOW CHARACTER
610
HAS SEND SYNC DATA TOKEN BEEN RECEIVED ?
611
NO
PROCESS SYNCHRONOUS DATA FROM OTHER MODULES
612
YES
TRANSMIT ALL SYNCHRONOUS PACKETS THEN PASS TOKEN TO NEXT MODULE
612
HAS END OF SYNCHRONOUS DATA CHARACTER BEEN RECEIVED ?
614
NO
PROCESS SYNCHRONOUS DATA FROM OTHER MODULES
615
YES
HAS ASYN DATA TOKEN BEEN RECEIVED ?
616
NO
PROCESS ASYN DATA FROM OTHER MODULES
617
YES
HAS ALL ASYNCHRONOUS DATA BEEN SENT OR HAS END OF ALLOCATED TIME ARRIVED ?
619
NO
SEND ASYNCHRONOUS DATA
620
YES
SEND ASYN DATA TOKEN TO NEXT MODULE
621
HAS SEND ASYNC DATA TOKEN OR START OF NEW WINDOW CHARACTER BEEN RECEIVED ?
622
NO
PROCESS ASYN DATA FROM OTHER WINDOWS
623
YES
HAS START OF NEW WINDOW BEEN RECEIVED ?
625
NO
YES

COMBINED SYNCHRONOUS AND ASYNCHRONOUS MESSAGE TRANSMISSION

RELATED APPLICATIONS

Applicant claims priority of applications application Ser. No. 09/268,099 which was filed Mar. 13, 1999, and application Ser. No. 60/098,297 filed Sep. 27, 1998. The present invention is a continuation of application Ser. No. 09/268,099 which was filed Mar. 13, 1999 and which is now pending and which was a continuation-in-part of application 60/098,297 filed Sep. 27, 1998.

BACKGROUND OF THE INVENTION

Packet switching systems transmit data by breaking the data into relatively small manageable pieces called packets. Packet switching can be used to transmit data in both computer networks and in telephone voice networks. Telephone packet switching networks transmit a series of packets over the same route in the network. Such systems in effect establish a virtual circuit from the point where a series of packets enters the network to the point where the packets are delivered. Packet switching networks establish virtual circuits through the network in order to transmit voice without delay and distortion.

Protocols such as the Internet ITPC protocol can transmit voice without establishing a virtual circuit connection, however, voice transmission using this type of protocol generally has less quality than voice transmitted using protocols which establish virtual circuits between the input point and the output point in the network..

Today, some voice transmissions are being made over packet protocols (such as the Internet) which do not establish virtual circuits. Voice connections over such circuits are of relatively low quality. The packet protocols which are used in the public telephone network are packet protocols which establish virtual circuits and which transmit all the packets that constitute a conversation over the same route through the network. Thus they provide high quality connections.

Data communication protocols can be characterized as either synchronous or asynchronous. Examples of widely used synchronous protocols are the X.25 protocol, and the frame relay protocol. Examples of widely used asynchronous protocols are the Ethernet, FDDI and ATM protocols. The X.25 protocol, the frame relay protocol and the ATM protocol are widely used in telephone systems. The Ethernet protocol and the FDDI ring protocol are widely used in local area networks (LANS) and wide area networks (WANS) that are used to interconnect computer systems.

There are various well known techniques for controlling asynchronous networks. One technique termed "carrier sense, multiple access with collision detection (CSMA/CD) is used in Ethernet networks. Another technique called token passing is used in FDDI ring networks.

Explanations of various synchronous and asynchronous protocols, and an explanation of CSMA/CD and FDDI ring networks is for example given in a book entitled "Voice and Data Communications Handbook" by Regis J. Bates and Donald Gregory which is published by McGraw Hill.

SUMMARY OF THE INVENTION

The present invention provides a ring protocol and system that combines synchronous and asynchronous transmission techniques. The ring can interconnect a number of modules

and be utilized to transmit both fixed and variable packets between the modules. Communication time is broken into a sequence of fixed length windows. At the beginning of each window the modules communicate using a synchronous protocol. That is, at the beginning of each window, if any unit has synchronous traffic, such traffic is transmitted using a synchronous ring protocol and fixed length packets. Virtual circuits can be established between the modules using the synchronous fixed length packets communicated at the beginning of each window. When it is desired to establish a virtual circuit between any of the modules in the ring, each module is assured that at the beginning of each window, space will be allocated to transmit a synchronous fixed length packet to another module in the ring. The windows occur frequently enough that a virtual voice grade circuit can be established between the modules. After all synchronous packets required during any window have been transmitted, asynchronous variable length data packets are transmitted around the ring. Limits are provided relative to the number of asynchronous packets any one module can transmit, thereby avoiding monopolization of the ring by any one module. The modules are synchronized by a periodically circulating a timing control character around the ring.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an overall block diagram of the preferred embodiment.

FIG. 2 is a diagram of the package assembler and disassembler (PAD) portion of the modules for coupling to analog telephone devices.

FIG. 3 shows a repeating sequence of fixed-length time windows used in allocating data traffic between the modules of FIG. 1.

FIG. 4 is a system-level flow chart illustrating overall operation of the communication system of FIG. 1.

FIG. 5 is a program flow chart showing the operation of the module which executes certain control among the modules of FIG. 1.

FIG. 6 is a flow chart showing the operation of modules of FIG. 1 when receiving and processing information exchanged there-between.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A preferred embodiment of the invention is shown in FIG. 1 and described herein. FIG. 1 shows a communication system 8 which supports integrated exchange of asynchronous and synchronous data between a number of modules 10*a*, 10*b*, etc. The data exchanged between the modules includes data traffic from and to computers 18 and voice traffic from conventional telephone devices 20 and 22. The data traffic from and to the computers 18 can include all of the various types of data traffic conventionally generated by computers such as video data, pc-phone data, etc. The system shown in FIG. 1 establishes "virtual circuits" between modules 10 for telephone traffic (i.e. synchronous transfer) and to also manages exchange of asynchronous information transfer between modules 10.

As shown in FIG. 1 system 8 includes a collection of modules 10 organized in a ring architecture. Information travels from one module 10 to a successive module 10 as required. The information on the ring can be divided into three categories, namely, data packets, tokens, and control characters (including a timing character). As used herein the following terms have the following meaning.

A "byte" consists of ten bits. Eight data bits are coded into a ten bit byte. Since a ten bit byte is used to encode 8 bits of data, a byte can be decoded into 256 different data words plus 768 additional decodes. Some of the additional decodes are used to form control characters, a timing character, and tokens. Other ones of the additional decodes are used for purposes unrelated to the present invention. Such decoding is conventional.

A "control character" consists one byte. The specific byte that forms each control character is selected from the decodes which do not form data words. There are three control characters which are used to implement the present invention. One control character indicates the start of a window, a second indicates the end of synchronous data transfer, and the third indicates the end of a window. When a module receives a control character it immediately retransmits the character to the next module in the ring.

A "token" consists of two bytes of data. As with control characters, the specific bytes that form each token are selected from the decodes which are not otherwise assigned. When a module receives a token, it only retransmits the token to the next module if certain conditions have been met. There are two tokens used in the implementation of the present invention. One token indicates that a module should begin transmitting its synchronous data and the second indicates that a module should begin transmitting asynchronous data. A module only passes a token to the next module after a module that receives a token has completed the task initiated by the token.

A "timing character" consists of one byte. This one byte is selected from the decodes not otherwise assigned.

A "synchronous packet" consists of 16 bytes of data plus three bytes of address information.

A "asynchronous packet" consists of 64 to 1524 bytes of data plus an 8 byte Ethernet header.

As each module **10** receives bytes (i.e. information packets, control characters, and tokens) from its predecessor in the ring, the module copies the bytes it receives internally and retransmits the bytes to a successive module **10** if appropriate. Bytes thereby flow at high speed around the ring architecture from one module **10** to another module **10**. It is noted that tokens are only transmitted from one module to another module when a task initiated by the token has been completed.

The ring architecture allows use of a variable number of modules **10**. Four such modules **10**, individually **10***a*–**10***d*, are shown in the particular embodiment described herein. It will be understood, however, that more modules **10** may be inserted into the ring architecture or that some of modules **10***a*–**10***d* may be removed from the ring architecture. Thus, modules **10** "stack" to meet use requirements, e.g., system **8** expands, to follow a growing user population or capacity requirement.

Each module **10** communicates with two adjacent modules through interconnecting communication links **12**. Each of the links **12** is bidirectional. For example link **12***a* goes from module **10***a* to **10***b* and link **12***ab* goes from module **10***b* to module **10***a*. In normal operation the system uses links **12***a*, **12***b*,**12***c*, and **12***d*. If one of these links is down (i.e. broken) the system automatically switches to links **12***ab* to **12***da*. Such use of a set of backup links is conventional.

Link **12***a* couples the output port **14***a* with the input port **16***b* of module **10***b*. Link **12***b* couples the output port **14***b* with the input port **16***c* of module **10***c*. Similarly, link **12***c* couples the output port **14***c* with the input port **16***d* of module **10***d*. Finally, link **12***d* couples the output port **14***d* with the input port **16***a* of module **10***a*. Each communication link **12** is a high-speed communication path. The capacity for links **12** is established depending on the number of modules **10** involved and the number of user devices attached to modules **10**. In the specific embodiment shown herein communication links **12** operate at 300 Mbps.

Modules **10** handle both synchronous data packets and asynchronous data packets. The synchronous packets **25** are fixed-length 16 byte packets **25**. The asynchronous packets are variable length packets **35**. Each module **10** has a number of asynchronous ports **30** (designated **30***a* to **30***d*) coupled to an asynchronous data buffer **32** and a number of synchronous ports **34** (designated **34***a* to **34***d*) coupled to a synchronous data buffer **36**. Computers **18** are connected to asynchronous data ports **30** and telephones **20** and **22** are connected to synchronous data ports **34**.

Modules **10** are interconnected by links **12**. Links **12** establish a combined synchronous and asynchronous message transmission data network whereby computers **18** may share resources and data and whereby telephone conversations may be conducted among the population of telephones **20** and **22**.

Each module **10** has a timer **37** (individually identified as timers **37***a* to **37***d*) which controls the timing within the module. Each module also includes a conventional programmed RISC processor and an associated memory which store and execute the programming operations described below.

Each module also has a plurality of user devices connected thereto. As shown in FIG. **1**, the user devices connected to the modules **10** include various computer work stations or terminals **18**, analog telephones **20**, and digital phones **22**.

Each analog telephone **20** is connected to a packet assembler and disassembler (PAD) **24**. FIG. **2** illustrates in more detail a PAD **24**. A PAD includes an analog-to-digital converter **24***a*, a sampling circuit **24***b*, a packet buffer **24***c*, and a packet assembler and disassembler block **24***d*. Each PAD **24** produces a sequence of 16 byte packets **25** representing voice sampled from a corresponding analog telephone **20**. Such a sequence of packets carry "one side" of a telephone conversation.

Each PAD **24** also receives a sequence of 16 byte packets **25** for audible presentation of voice at the corresponding analog telephone **20**. Block **24***d* drives a digital-to-analog converter **24***e* with incoming packets **25**, i.e., the "other side" of a telephone conversation involving a user and telephone **20**. Thus, block **24***d* operates within a given module **10** providing and receiving packets **25** representing an analog telephone conversation and the associated normal inband telephone signals. Packet **25** transport occurs at 64 kbps in order to support the full duplex telephone traffic.

Digital telephones **22** produce similar packets **25** representing one side of a telephone conversation, i.e., voice sampled by digital telephone **22**, and also receive a sequence of packets **25** for audible presentation at a digital telephone **22**. Digital telephones **22** exchange packets **25** with a module **10** at sufficient speed to support a full duplex telephone conversation, i.e., 64 kbps.

Telephone conversation data from telephones **20** and **22** is packetized in the fixed-length 16 byte packets **25**. The packets containing voice data must be delivered in a timely manner in order to maintain acceptable quality of voice communication. In order to accomplish timely delivery of data representing voice communications, all such data is handled by the present invention in a synchronous fashion.

Since such data is handled in synchronous fashion conventional "virtual circuits" can be established between user devices, e.g., between members of the population of telephones **20** and **22**. High quality telephone connections can therefore be achieved. There is no perceptible degradation in voice quality because no more than about a four and a half millisecond delay exists in delivery of any given packet **25** from end terminal to end terminal (i.e. from telephone to telephone).

Computer work stations **18** produce data for delivery to other stations **18** and receive data from other stations **18**. Because variation in delay and variation in packet size is generally acceptably in communications between stations **18**, such data is managed in an asynchronous fashion when transported via modules **10**. Information exchanged among stations **18** is divided into "Ethernet" type packets, i.e., variable sized packets including addressing information according to an Ethernet type addressing schemes.

The time frame for communication on links **12** is divided into a sequence of windows. FIG. **3** illustrates a sequence of windows **50**, individually identified as windows **50***a*, **50***b*, etc. Each window **50** is two millisecond long

Each window **50** begins with a "start window" control character or field **52** (which is one byte long). The start window control character indicates the onset of a window **50**. The remainder of each window **50** is dedicated first to all pending synchronous data transmissions and then to asynchronous data transmissions. More particularly, a first portion **54** of each window **50** is dedicated to exchange of all pending synchronous data packets **25**. After all pending synchronous data packets **25** have been exchanged among modules **10**, a second control character (not explicitly shown in FIG. **3**) is transmitted around the ring to indicate the end of the synchronous transmissions. A second portion **56** of each window **50** is dedicated to exchange of asynchronous data packets **35**. At the end of each window another control character (not explicitly shown in FIG. **3**) is transmitted around the ring. As will be explained later, tokens and timing characters are also transmitted around the ring.

The length of window **50** is two milliseconds long. The length of windows **50** is established by taking into account the bandwidth of the various communication Is paths. The length of window **50** is established so that all synchronous data can be delivered during each window and so that after the synchronous data is transmitted, sufficient reserve will remain in each window **50** to conduct exchange of asynchronous data. The actual allocation of a given window **50** between synchronous and asynchronous data is dynamic. The allocation depends on the amount of pending synchronous data packets **25** which must be transmitted during the first portion of a given window **50**. As the number of telephone conversations increases, the portion **54** of window **50** used for such conversations increases.

Thus the allocation between synchronous and asynchronous data in a given window **50** is not fixed but rather a function of the amount of synchronous data pending at the beginning of the window **50** with the remaining portion **56** being used for asynchronous data. A control character **53** which indicates that synchronous traffic is "all done" separates portions **54** and **56** of each window. This control character indicates the end of synchronous data transmission and the beginning of asynchronous data transmission within a given window **50**.

FIG. **4** illustrates, at a system level, data exchange during a given window **50**. As shown in FIG. **4**, processing loops at decision block **100** until start window control character **52** appears on links **12**. Processing then advances to block **102** where modules **10** exchange all pending synchronous data packets, i.e., deliver all pending packets **25** in the synchronous data buffers **36**. Block **102** represents the overall exchange of all pending synchronous data packets **25** among modules **10***a*–**10***d* by ring message exchange.

As indicated by decision block **106** a determination is made that all modules **10** have completed exchange of pending synchronous data, e.g., all modules **10** have delivered all pending synchronous data packets **25**. In other words, all synchronous data in buffers **36** at the onset of the current window **50** have been transmitted via links **12** to an appropriate module **10**. At this point, communication among modules **10** switches from a synchronous mode of operation to an asynchronous mode of operation allowing variable length packets and an alternate addressing scheme. Asynchronous transmission is done using Ethernet packet rules and addressing codes to route the variable length packets to particular modules **10** and to corresponding user devices attached thereto. Ethernet packaging rules allow packets of varying length between 64 and 1524 bytes.

To prevent monopolization of window **50** by one module, each module **10** limits its use of portion **56** of each window so as to allow other modules **10** to deliver asynchronous data. Thus, block **108** represents delivery of a limited amount of asynchronous packets followed in decision block **110** which tests use limitations. During block **108**, a module **10** sends a certain number of Ethernet packets to a successive one of modules **10**. Such module **10** limits its further use of the asynchronous portion **56** of a given window **50**. That is, each module **10** is allowed a limited number of asynchronous data bytes per given window **50**. In the embodiment described herein, each module **10** transmits a maximum of 4000 bytes of asynchronous data in any given window **50**. Thus, system level operation loops at blocks **108** and **110** until all modules **10** have reached their use limit for the current window **50** or have delivered all pending asynchronous data. Processing eventually returns to block **100** where system **8** waits for occurrence of the start window control character **52** and a next window **50**.

The modules **10** in general operate on a "peer" basis. However, one of modules **10** is given some degree of control over the process. In the embodiment shown, module **10***a* executes master control, that is, to some extent module **10***a* orchestrates the exchange of information on links **12** and it is in effect a timing master for the system. Modules **10** other than module **10***a* may be inserted and removed from the system as needed or desired without corrupting an overall control strategy. However, there must always be a control module **10***a*. Master control module **10***a* makes use of control characters and tokens to orchestrate packetized information exchange within system **8**. Modules detect the receipt of a token or control character by detecting one of the decodes of a byte other than the 256 data decodes. When a module **10** receives a control character or a timing character, it immediately retransmits the control character or timing character to the next module **10**. When a module **10** receives a token, the token is held until the module **10** is ready to retransmit the token, i.e. until the module is ready to relinquish its right to send packets of a particular type.

FIG. **5** illustrates programming with respect to operation of module **10***a*. Module **10***a* is the master control module. As indicated by block **510**, the process begins when module **10***a* transmits a "window start" control character **52** (see FIG. **3**). Control character **52** is sent immediately around the ring architecture because when a module **10** receives a control character it immediately re-transmit (i.e. repeats) the control

character. At this point, all modules **10** are prepared for the onset of a window **50**. As indicated by block **511** module **10***a* transmits all its pending synchronous packets **25**, i.e., module **10***a* empties synchronous data buffer **36***a*, into the ring on communication link **12***a*. Once module **10***a* has transmitted all of its synchronous data packets **25**, as indicated by block **512** module **10***a* passes the "transmit synchronous packets" token to the next module, i.e., to module **10***b*.

After passing the transmit synchronous packets token to the next module on the ring, module **10***a* processes synchronous packet traffic from other modules until the transmit synchronous packets token is returned to module **10***a*. This is indicated by blocks **513** and **514**. During this time, the remaining modules **10** will each in turn have opportunity to send all synchronous data packets **25** which were pending at the onset of the current window **50**. That is, after module **10***b* receives the token, it transmits all its pending synchronous data packets **25**, i.e., empties synchronous data buffer **36***b*, onto link **12***b*. Module **10***b* then passes the token to module **10***c*, giving module **10***c* an opportunity to send all its synchronous data packets **25** onto link **12***c*. The token is then passed to module **10***d*. When module **10***d* receives the token, it in turn submits all its synchronous data packets **25** which were pending at the onset of the current window **50** onto link **12***d*. As each module **10** submits its synchronous data packets **25** onto the ring architecture, each packet **25** reaches a target or module **10** which has attached thereto one of telephones **20** or **22** so as to complete a virtual circuit. Eventually, all modules **10** will have had an opportunity to submit synchronous data packets **25** onto the ring and the "transmit synchronous packets" token will return to module **10***a*. Processing then advances from decision block **513** to block **516** and module **10***a* sends the "all done synchronous data" control character **53** out on link **12***a*. Each of modules **10***b*–**10***d* thereby receive the "all done synchronous data" control character **53** indicating a transition from synchronous data exchange to asynchronous data exchange. As indicated by blocks **517** and **518**, module **10***a* transmits asynchronous data packets **35** onto link **12***a*. As may be appreciated, each of the asynchronous data packets pass around the ring and eventually reach the intended destination, i.e., one of modules **10***b*–**10***d* addressed as the destination address in the packet.

Module **10***a* stops sending asynchronous data packets when one of two conditions is satisfied as indicated by decision block **517**. Transmission of asynchronous packets by module **10***a* stops when module **10***a* determines that it has no more asynchronous data to transmit, (i.e., asynchronous data buffer **32***a* is empty) or if module **10***a* has reached its use limit. In defined a module is limited to transmitting 4000 bytes in one session. If module **10***a* has not reached its use limit and if there are additional asynchronous data packets **35** in buffer **32***a*, then processing returns to block **518** and module **10***a* continues to transmit asynchronous data packets. Eventually, module **10***a* either reaches its use limit or exhausts pending asynchronous data in buffer **32***a*. Processing then advances to block **520** and module **10***a* passes the token to the next module, i.e., to module **10***b*.

After passing the token to the next module, module **10***a* will process asynchronous data traffic as indicated by blocks **521** and **522**. Module **10***a* checks for return of the token as indicated by decision block **421**. That is processing as indicated by blocks **521** and **522** continues until the token is returned to module **10***a*. When the token returns to module **10***a*, all modules **10** have had opportunity to send asynchronous data packets **35** onto the ring at least once up to their given limit, i.e., at least 4000 bytes.

At this point, some portion of window **50** may remain. This is determined as indicated by block **523**. If more time remains module **10***a* can take advantage of this opportunity to send more asynchronous data packets **35**. As indicated in FIG. **5** The "no" output from block **523** goes back to block **517** and the process repeats.

Eventually, module **10***a* runs out of time in the current window **50** for transmission of additional packets **35**. An end of ring control character is then transmitted around the ring. Processing then returns to block **510** where module **10***a* again sends control character **52** and the process repeats.

Module **10***a* also provides a timing reference for the system. Module **10***a* includes an interval timer **37***a* which produces interrupt signal every 15.625 microseconds. This timing reference signal is transmitted from module **10***a* to the other modules in the ring. When the timing interrupt occurs, module **10***a* transmits the special timing control character. The timing control character is inserted into any packet that is being transmitted at the time the interrupt occurs. Thus, some synchronous packets **25** may be 17 bytes long after the clock control character is inserted therein. The additional delay introduced, i.e., a 16 byte synchronous packet **25** versus a 17 byte synchronous packet **25**, does not introduce any noticeable delay to persons engaged in a conversation. The timing character is a 10 bit character which is not used for any other purpose and which each units recognizes as a timing character. When a unit on the ring (other than module **10***a*) detects this character, it repeats the character to the next unit on the ring and at the same time it re-synchronizes its internal clock **37**. That is, the clock **37** in each unit is re-synchronized when the timing character is detected. In this way the clocks **37** in the various modules are kept in close synchronization. It is noted that as shown herein it is the control module **10***a* which introduces the timing character onto the ring, any one of the modules could perform this function.

FIG. **6** illustrates the programming for modules **10** other than module **10***a*. (FIG. **5** shows the programming for module **10***a*) That is, FIG. **6** illustrates programming for modules **10***b*–**10***d*. The process starts as indicated by block **610** when a module receives a "start window" control character **52**. After receiving a "start window" control character a module looks for a token with indicates that the module should start transmitting synchronous data. Processing continues iteration between blocks **611** and **612** until the module **10** receives the "start transmitting synchronous data" token. As indicated by block **611**, when a module receives the "start transmitting synchronous data" token the module begins transmission of its synchronous data. Once a module **10** has transmitted all pending synchronous data packets **25** into the ring, it passes the "start transmitting synchronous data" token to the next module **10**. This gives the next module **10** opportunity to submit its synchronous data packets **25**.

Once a module **10** has passed the "start transmitting synchronous data" the module processes synchronous packet traffic from other modules as indicated by block **615**. Decision block **614** indicates that a module looks for "all synchronous traffic done" control character **53**. Until the "all synchronous traffic done" control character **53** appears, processing iterates at blocks **614** and **615** and the module processes any synchronous data packets **25** appearing in the ring architecture from other modules. When the "all synchronous traffic done" control character **53** does appear processing advances to block **616** and **617** where the module processes any asynchronous packet traffic appearing in the ring architecture from other modules. In decision block **616**,

module **10** looks for the "send asynchronous data" token. Processing iterates at blocks **616** and **617** until the module receives the "send asynchronous data" token. "send asynchronous data" token is received asynchronous data packet **35** are transmitted as indicted by block **620**. After transmitting each asynchronous data packet **35**, the module determines as indicated by block **619** whether it has any additional asynchronous data packets to transmit, or if the module has reached its use limit, e.g., has transmitted 4000 bytes of asynchronous information in the current window **50**. Processing iterates at blocks **619** and **620** until no further asynchronous data packets remain or until the module **10** has reached its allotment or use limit allowed in the current window **50**. Processing then advances to block **621** and the module passes the "send asynchronous data" token to the next module **10**.

After each of the modules **10** has had an opportunity to transmit both synchronous and asynchronous traffic during a particular window, there may still be time remaining in the window. When this condition occurs, the modules are given another chance to transmit additional asynchronous packets. This condition is illustrated in FIG. **6** by the path from block **622** through block **625** to the entry of block **619**.

Following block **621**, processing iterates between blocks **622** and **623** where the module processes any further asynchronous packet traffic from other modules and looks for the occurrence of the end of window and the start new window control characters. If an end of window and start of new window control characters have not been received the processing goes from block **622** to **625** to **619**. When and end of window and start of new window control characters are received the processing returns to block **611**

As an example of the capacity of the system, it is noted that a voice switching capacity on the order of 256 simultaneous, full duplex calls may be implemented on a stack of eight modules **10**. Each full duplex voice call consumes 64 kbps of data bandwidth. This translates into: (64,000*2)*256=32,768,000 or 32 Megabits (Mbps) of voice switching bandwidth on links **12** to support 256 simultaneous full duplex voice calls. This is exclusive of framing overhead, which will be dependent on the hardware implementation.

There are also other capacity considerations. Services which use 'redirection', e.g., voice compression, voice recognition or fax services, all are very processor bandwidth intensive. This component creates an 'overhead factor' on the base voice switching bandwidth independent of framing overhead. Using an estimated overhead factor of thirty-three percent, the voice switching bandwidth requirement increases from about 32 Mbps to about 44 Mbps.

For asynchronous data capacity, a minimum carrying requirement of a single 100 Mbps Ethernet will meet a given level of computer network use expectations. This brings the aggregate carrying capacity for system **8** to about 144 Mbps. For control and management an overhead of not more than about 4 Mbps per module **10** is expected. Of this, 1 Mbps is reserved for true inter-module **10** communication, 0.5 Mbps is reserved for network management, 1.5 Mbps is reserved for call accounting and 1 Mbps is reserved for event logging and tracing. For eight modules **10** in a system **8**, a subtotal of (8*4,000,000) or about 32 Mbps. This brings the entire switching capacity requirement for system **8** to about 176 Mbps. Establishing a 300 Mbps capacity for links **12** supports this expected switching capacity.

Naturally, other systems could be implemented using the present invention with different requirements and capacity considerations. For example various numbers of modules **10** could be connected in a ring utilizing the present invention. While the invention is described as applied to a LAN environment, it is noted that the invention could also be applied in a WAN environment. It is also noted that the communication path between modules **10** could be either electrical or optical without departing from the present invention.

It should be appreciated that the computer data communicated between modules **10**, could include all the various kinks of data normally transmitted between computer. For example such data could include video data and PC-telephone data.

A combined synchronous and asynchronous message transmission method and apparatus has been shown and described. The integration of synchronous and asynchronous message transmission into a single communication system provides opportunity for integrated communication services incorporating both computer data and voice data. Despite integration of synchronous and asynchronous data, synchronous data arrives in timely fashion without degradation in voice quality.

As described herein data from computer **18** is treated as asynchronous data. However, under certain conditions, computers **18** may be required to deliver time-sensitive information in a synchronous manner, and could be treated as such by a corresponding module **10**.

System **8** also includes an external connection **38**, e.g., a high speed telephone or network connection, whereby other systems may introduce information into system **8** or take information from system **8**. Such links are handled similar to links directly connected to modules **10**.

It is noted herein that a single asynchronous protocol is used. However, additional control schemes may also be employed. For example such protocols could be used to allow further exchange of asynchronous data when modules **10** have reached their use limit for asynchronous data, but window **50** has not yet expired.

It will be appreciated that the present invention is not restricted to the particular embodiment that has been described and illustrated, and that variations may be made therein without departing from the scope of the invention as found in the appended claims and equivalents thereof.

What is claimed is:

1. A system including,

a plurality of modules,

a communication channel coupling said modules together in a ring configuration, operation of said communication channel being divided into repeated fixed length windows, each fixed length windows being divided into a variable length first portion and a variable length second portion,

fixed length packets being transmitted between said modules during said first portion of each of said fixed length windows and variable length packets being transmitted between said modules during said second portion of each of said repeated fixed length windows,

each of said modules including means for providing and receiving synchronous information in substantially fixed length packets during said first portion of each of said fixed length windows, means for providing and receiving asynchronous information in variable length packets during said second portion of each of said fixed length windows, each of said packets comprising a plurality of multi bit bytes; means for transmitting a

first token to the next module in the ring when all available fixed length packets up to a maximum number have been transmitted during said first portion of a fixed length window, and means for transmitting a second token to the next module in the ring when all available variable length packets up to a maximum number have been transmitted during said second portion of a fixed length window,

means for switching from the transmission of synchronous packets to asynchronous packets when said token traverses said entire ring,

whereby both synchronous and asynchronous information can be transmitted between said modules using said communication channel.

2. A system comprising:

a plurality of modules, each of which includes means for providing and receiving synchronous information in substantially fixed length packets and for providing and receiving asynchronous information in variable length packets;

a communication channel and associated protocol coupling said modules in a ring configuration, said protocol including repeating fixed length windows and allowing during a first portion of each of said windows exchange of said synchronous information pending at the onset of said each of said windows and allowing during the remaining portion of said each of said windows exchange of at least a portion of said asynchronous information, and

means for transmitting a token between modules whereby each module can pass said token to the next module when it completes sending synchronous packets and said system begins sending asynchronous packets when said token traverses said entire ring.

3. A system according to claim 2 wherein the capacity of said communication channel is sufficient in relation to an expected magnitude of said information to be exchanged among said modules and in relation to a duration of said each frame to ensure complete exchange of said synchronous information whereby any given item of synchronous information takes no longer than said duration to travel from a source to a destination.

4. A system according to claim 3 leaving a sufficient remainder of said each window following said complete exchange to exchange enough of said asynchronous information to meet an expected capacity for said asynchronous information.

5. A system according to claim 2 wherein said communication modules are organized in a ring architecture and exchange information according to a token passing method.

6. A system according to claim 2 wherein at least one of said communication modules couples to a plurality of user devices, a first portion of said plurality of user devices interacting with said at least one of said communication modules by exchange of synchronous information.

7. A system according to claim 2 wherein each module includes a clock, and wherein synchronization is maintained between said modules by periodically transmitting a character on said ring which is recognized by each module as a timing character and wherein each modules resynchronizes its clock when said character is received.

8. A system which utilizes synchronous and asynchronous transmission methods, said system comprising:

a plurality of modules coupled in a ring architecture, each module receiving input from a predecessor module and providing output to a successor module, each of said modules referencing a repeating sequence of time windows, each time window including a first portion dedicated to exchange of synchronous data and a second portion dedicated to exchange of asynchronous data, and each module including means for transmitting a token to the next module in the ring when all available synchronous packets up to a maximum number have been transmitted during the first portion of a window

whereby said system can switch from transmitting synchronous packets to transmitting asynchronous packets when said token has traversed said entire ring.

9. A system according to claim 8 wherein said information exchange is conducted according to a token passing method.

10. A system according to claim 8 wherein said system establishes virtual circuits between user devices coupled to said modules, said virtual circuits delivering said synchronous data during said first portion of each of said time windows.

11. A system according to claim 8 wherein said synchronous data is broken into fixed length packets and said asynchronous data is broken into variable length packets.

12. A system according to claim 8 wherein each of said modules limits use of said second portion of each of said windows whereby each of said modules has opportunity during said second portion of said window to transmit asynchronous data.

13. A system according to claim 8 wherein each module includes a clock, and wherein synchronization is maintained between said modules by periodically transmitting a character on said ring which is recognized by each module as a timing character and wherein each modules resynchronizes its clock when said character is received.

14. A system according to claim 8 wherein said first portion terminates when all pending synchronous data has been exchanged among said plurality of modules.

* * * * *

US005935214A

United States Patent [19]

Stiegler et al.

[11] Patent Number: 5,935,214

[45] Date of Patent: Aug. 10, 1999

[54] METHOD FOR TRANSMITTING SOURCE DATA AND CONTROL DATA IN A COMMUNICATION SYSTEM WITH A RING STRUCTURE

[75] Inventors: Andreas Stiegler, Ettlingen; Patrick Heck, Durmersheim; Herbert Hetzel, Schwaigen; Hans-Peter Mauderer, Gaggenau, all of Germany

[73] Assignees: Silicon Systems GmbH Multimedia Engineering, Karlsruhe; Becker GmbH, Karlsbad, both of Germany

[21] Appl. No.: 08/949,724

[22] Filed: Oct. 14, 1997

[30] Foreign Application Priority Data

Oct. 11, 1996 [DE] Germany 196 42 258

[51] Int. Cl.[6] H04J 3/04

[52] U.S. Cl. 709/238; 709/251; 370/389; 370/330; 370/351

[58] Field of Search 395/200.81, 200.68; 370/389, 330, 351; 709/251, 238

[56] References Cited

U.S. PATENT DOCUMENTS

4,958,344	9/1990	Scott	370/535
4,987,572	1/1991	Scott	370/538
5,361,261	11/1994	Edem et al.	370/445
5,490,168	2/1996	Phillips et al.	375/224

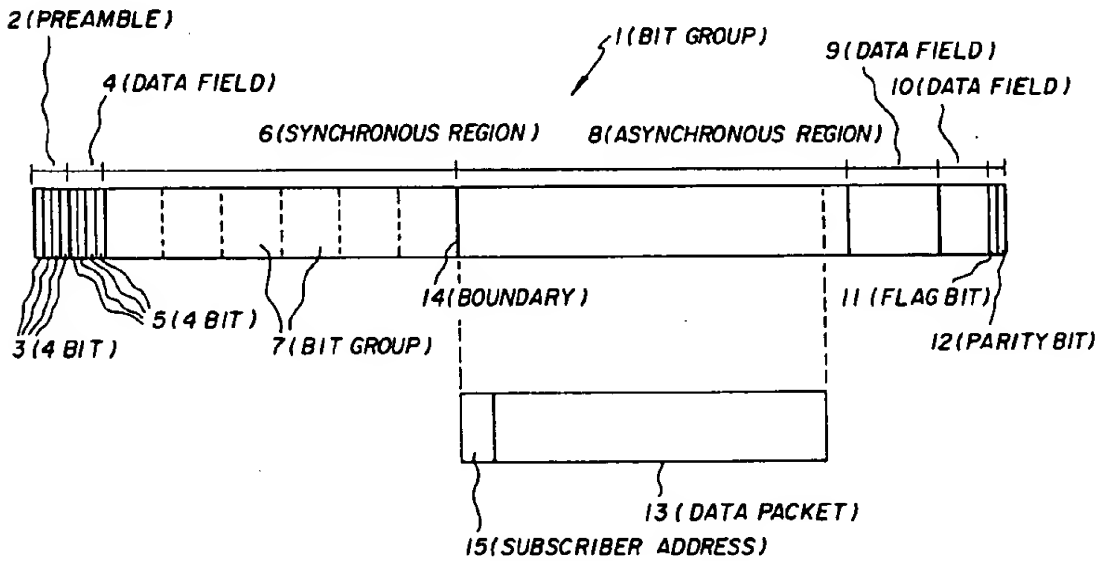
Primary Examiner—Krisna Lim

Attorney, Agent, or Firm—Herbert L. Lerner; Laurence A. Greenberg

[57] ABSTRACT

A method for the common transmission of digital source data and control data between data sources and data sinks. The data sources and data sinks are subscribers in a communication network with a ring structure. The source data and control data are transmitted in a format which prescribes a pulsed sequence of individual bit groups of identical length. Specific bit positions in each of the bit groups are reserved for source data and control data. The transmission is in a continuous data stream synchronous with a clock signal. An arbitrarily large contiguous region of the bit positions can be reserved for the source data within a bit group for data which are transmitted in data packets. Each of the data packets has a start with a subscriber address and a defined length.

27 Claims, 6 Drawing Sheets



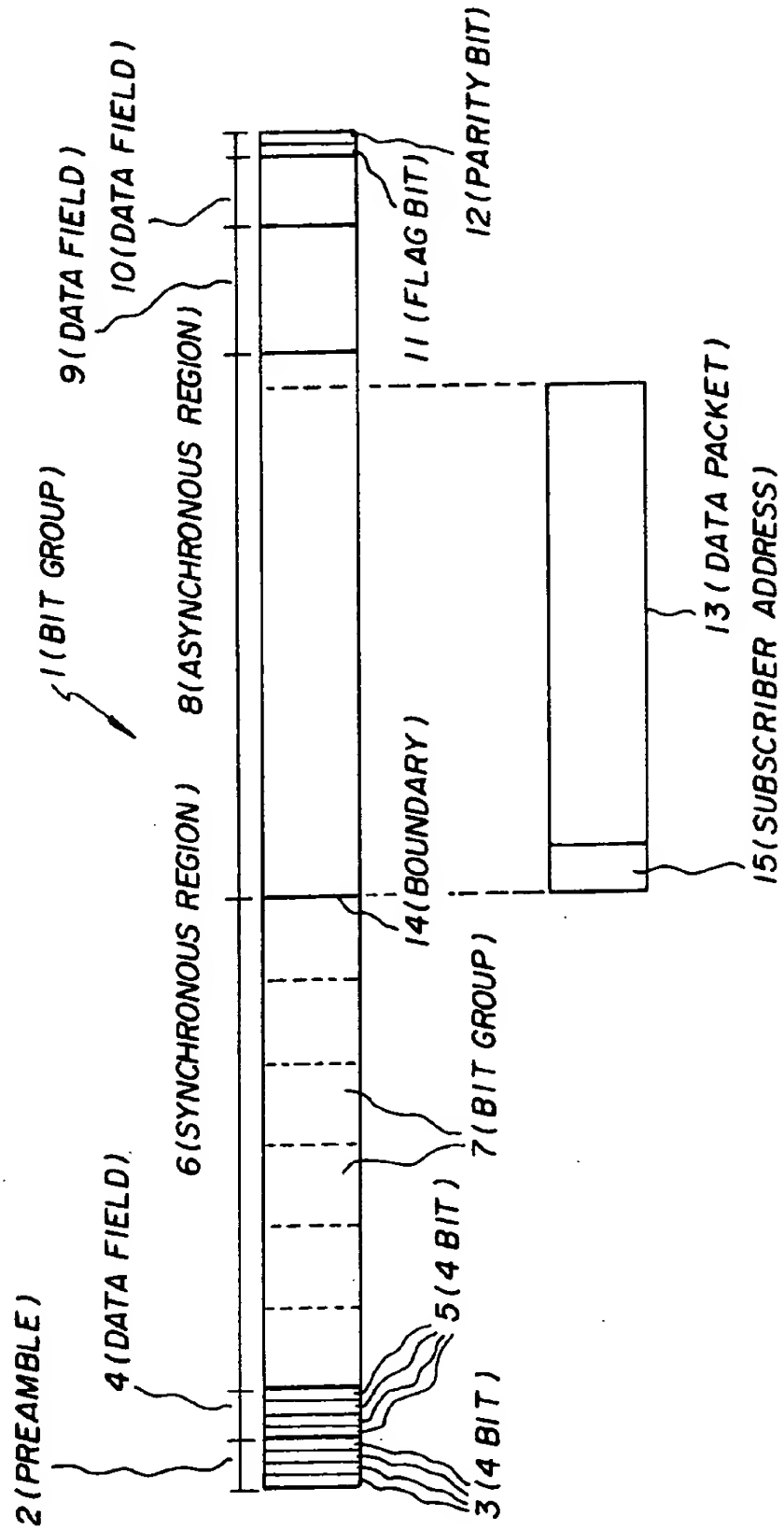

1(BIT GROUP)
2(PREAMBLE)
3(4 BIT)
4(DATA FIELD)
5(4 BIT)
6(SYNCHRONOUS REGION)
7(BIT GROUP)
8(ASYNCHRONOUS REGION)
9(DATA FIELD)
10(DATA FIELD)
11(FLAG BIT)
12(PARITY BIT)
13(DATA PACKET)
14(BOUNDARY)
15(SUBSCRIBER ADDRESS)

Fig. 1

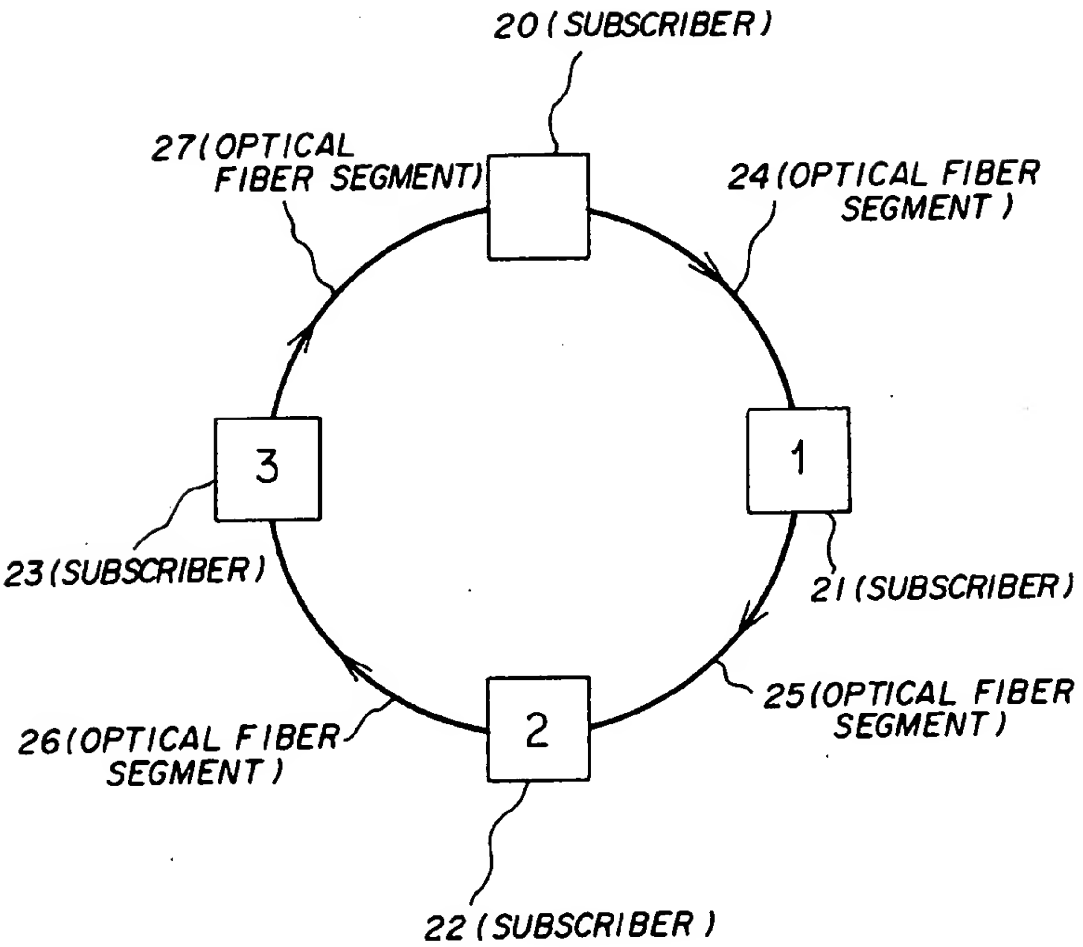

20 (SUBSCRIBER)
27 (OPTICAL FIBER SEGMENT)
24 (OPTICAL FIBER SEGMENT)
1
3
23 (SUBSCRIBER)
21 (SUBSCRIBER)
25 (OPTICAL FIBER SEGMENT)
26 (OPTICAL FIBER SEGMENT)
2
22 (SUBSCRIBER)

Fig. 2

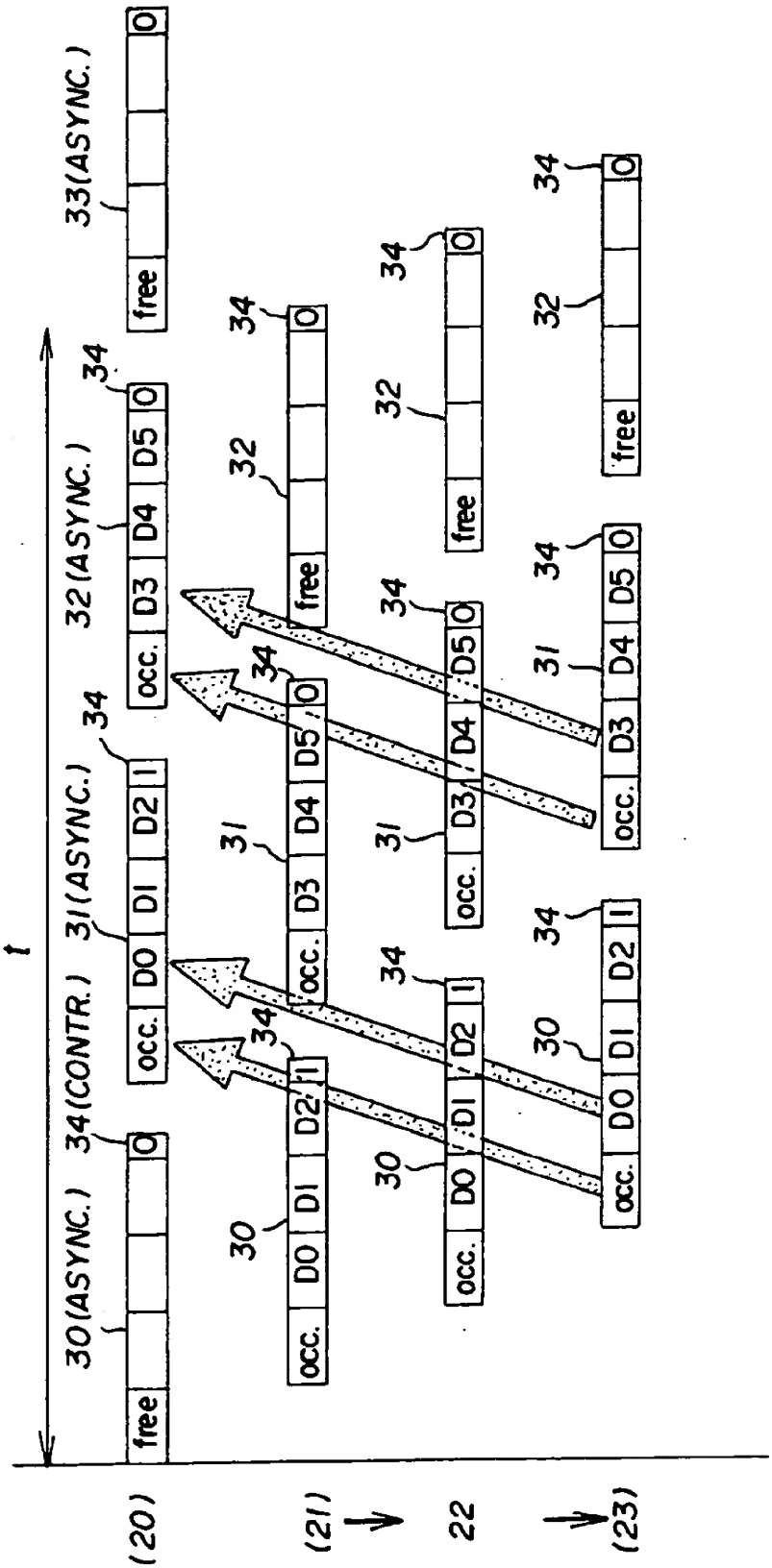

t
30(ASYNC.)
34(CONTR.)
31(ASYNC.)
32(ASYNC.)
33(ASYNC.)
free
occ.
D0
D1
D2
D3
D4
D5
0
1
30
31
32
34
(20)
(21)
22
(23)

Fig. 3

Fig. 4

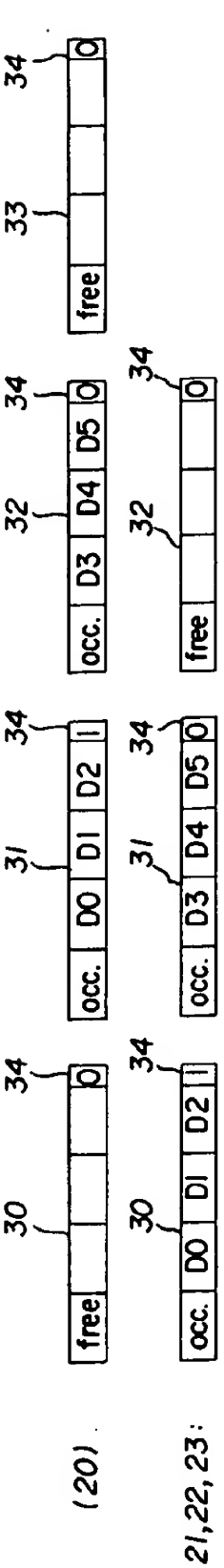
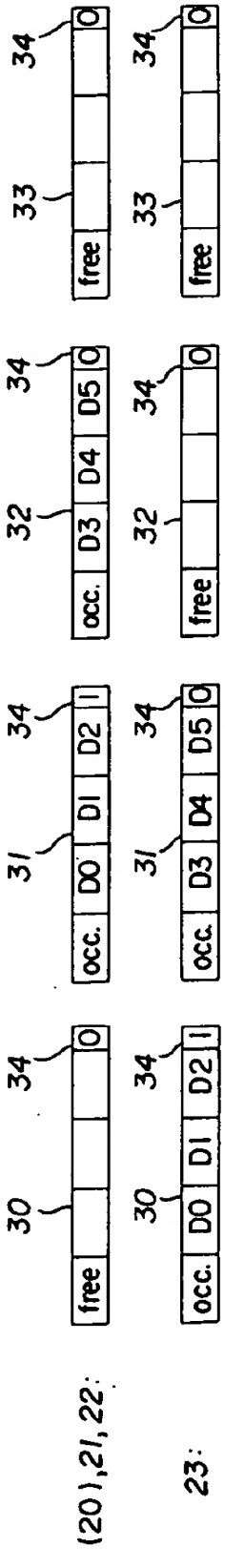

(20)
21,22,23:
free
occ.
D0
D1
D2
D3
D4
D5
30
31
32
33
34
0
1

Fig. 6


(20),21,22:
23:
free
occ.
D0
D1
D2
D3
D4
D5
30
31
32
33
34
0
1

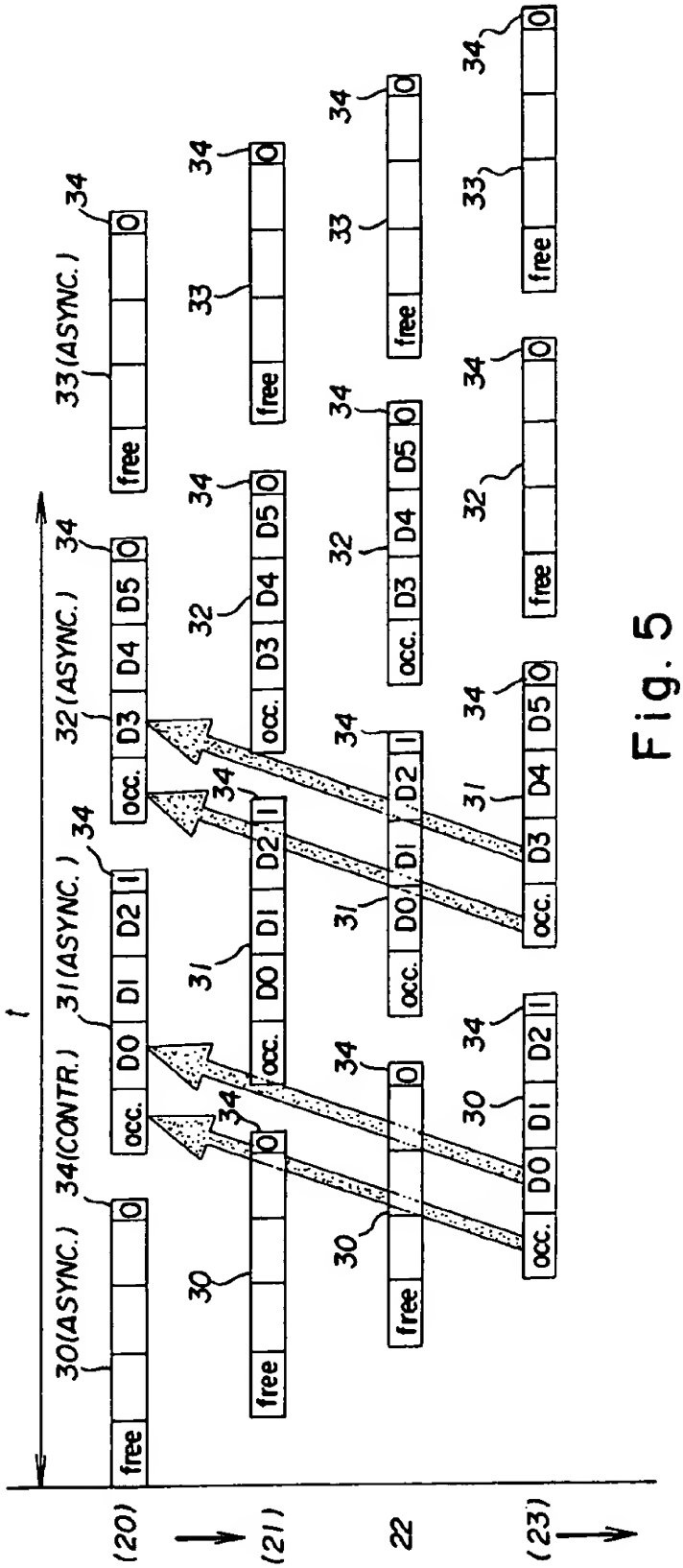

t
30(ASYNC.)
34(CONTR.)
31(ASYNC.)
32(ASYNC.)
33(ASYNC.)
free
occ.
D0
D1
D2
D3
D4
D5
0
1
30
31
32
33
34
(20)
(21)
22
(23)

Fig. 5

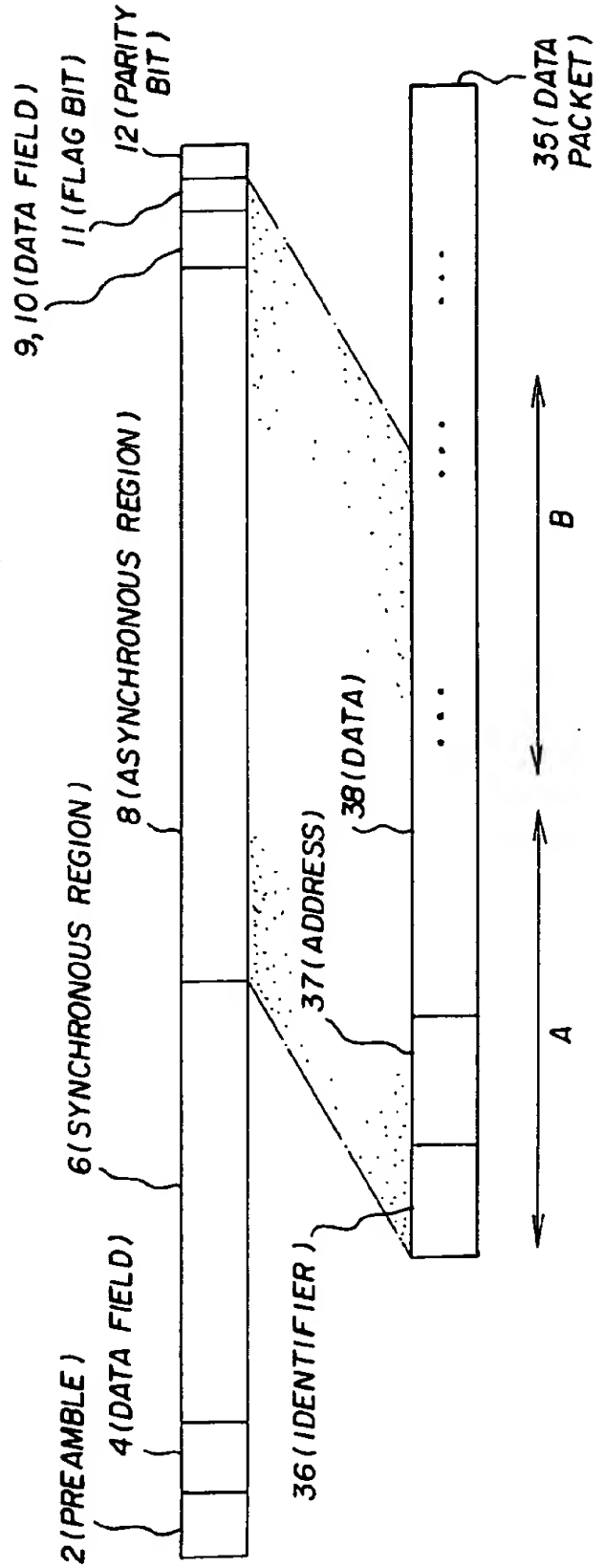

2(PREAMBLE)
4(DATA FIELD)
6(SYNCHRONOUS REGION)
8(ASYNCHRONOUS REGION)
9,10(DATA FIELD)
11(FLAG BIT)
12(PARITY BIT)
35(DATA PACKET)
36(IDENTIFIER)
37(ADDRESS)
38(DATA)
A
B

Fig. 7

METHOD FOR TRANSMITTING SOURCE DATA AND CONTROL DATA IN A COMMUNICATION SYSTEM WITH A RING STRUCTURE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to a method for the common transmission of digital source data and control data between data sources and data sinks, which are subscribers to a communication network with a ring structure, the source data and control data being transmitted in a format which prescribes a pulsed sequence of individual bit groups of identical length in which specific bit positions are reserved in each case for source data and control data, which are transmitted in a continuous data stream synchronous with a clock signal. The clock signal is generated by a single subscriber. All other subscribers are synchronized with this clock signal. Data transmission methods which are purely packet orientated such as, for example, the ATM method, i.e., asynchronous transfer methods, are to be distinguished from these. The invention is also directed towards specific applications of the novel method in a domestic communication system and a communication system for motor vehicles.

Methods of this type are used wherever electric and electronic devices of different types which are intended to exchange information with one another are mutually networked, often in a complicated way, by means of data lines. Thus, for example, in audio applications such a method can be used to control communication between mutually networked data sources, on the one hand, such as CD players, radio receivers and cassette recorders, for example, and data sinks connected thereto, on the other hand such as, for example, amplifier/loudspeaker combinations. Certain devices can thereby be a data source and a data sink (e.g. cassette recorder/player).

2. Description of the Related Art

It has become known heretofore from European patent disclosure EP-A-0 725 522 to interconnect different types of network subscribers by optical fibers in such a way that the data stream passes each subscriber sequentially. An optical communication network with a ring structure is then produced which has particular advantages, in particular for mobile applications, for example in motor vehicles, and domestic applications, for example in multimedia networks. In order to be able to transmit data between a multiplicity of interconnected subscribers in such a network, the positional region within a bit group which is reserved for the source data transmitted in a continuous data stream can be subdivided into a plurality of component bit groups of identical length. It is thereby possible for the source data allocated to each component bit group to be assigned to a specific subscriber as a function of the control data. The component bit groups form channels which are available in each case to a specific subscriber for an unspecified time.

Data transmission in a data stream synchronous with a clock signal, as it is typical of the methods mentioned, permits simple connection to data sources and data sinks, which likewise transmit and receive continuously. This is the case with many audio and video devices, for example. In addition, the current quality requirements in the audio field, for example, can be met with acceptable outlay generally only given synchronous data transmission.

In principle, those methods can also be used to transmit data which originate from a device which supplies data in an asynchronous mode such as, for example, a fax machine or a CD-ROM drive. It is necessary for that purpose to synchronize the data delivered in bursts with the clock signal, and to transmit the synchronized data via a channel which is formed by specific bit positions and is assigned to the receiver of the data. During the time in which it is assigned to the receiver, the channel is not available for any other data. Since the assignment and rerelease of channels takes a relatively long time, a substantial amount of transmission capacity is sacrificed. Further transmission capacity is sacrificed by short gaps between individual bursts, which do not permit a channel release in the meantime.

The above-noted EP-A-0 725 522 proposes to transmit fax data or other unformatted, i.e., uncoded, data via so-called transparent channels which are provided in the regions reserved for the control data. These channels, however, are then available only for a very specific purpose, and the above-mentioned disadvantages are encountered. Again, the transmission capacity of the permanently reserved transparent channels is limited by virtue of the system.

SUMMARY OF THE INVENTION

It is accordingly an object of the invention to provide a method for data transmission in a communication network with a ring structure, which overcomes the above-mentioned disadvantages of the prior art devices and methods of this general type and which permits efficient transmission both of data transmitted continuously and of data transmitted in bursts.

With the foregoing and other objects in view there is provided, in accordance with the invention, a method for the common transmission of digital source data and control data between data sources and data sinks, which are subscribers to a communication network with a ring structure. The method comprises the steps of:

- transmitting source data and control data in a continuous data stream synchronized to a clock signal and in a format which prescribes a pulsed sequence of individual bit groups of equal
- length in which specific bit positions are reserved for source data and control data;
- defining data packets each having a start and a defined length, and carrying a subscriber address;
- reserving an arbitrarily large contiguous region of the bit positions for the source data within a bit group and transmitting the data packets within the contiguous region.

In other words, the objects set for the method are satisfied by allowing an arbitrarily large contiguous (coherent) region of the bit positions to be reserved for the source data within a bit group for data which are transmitted in data packets which each have a start and a defined length and which are assigned a subscriber address.

The size of the region for the data which are transmitted in data packets can be set in dependence on the requirement of the respective application. For example, in a communication system which contains only continuously operating data sources, there is initially no need at all to provide space for data in packets. If a fax machine or a CD-ROM drive, for example, is added to the system, this device will have available a sufficiently large region of bit positions for the source data present in packets. If the system has a plurality of such sources added, less than the sum of the individual space requirements generally needs to be reserved, since the data packets of one source which are identified by a unique subscriber address can be transmitted to another source by

using gaps between the data packets. This fundamental advantage of packet-orientated transmission methods is therefore achieved even in the case of the transmission method according to the invention, which is fundamentally synchronous.

A source of data packets writes successive bits of a data packet to be transmitted preferably in mutually adjacent bit positions which are reserved for the data that are transmitted in data packets. In this case, these data are synchronized with the clock pulse of the communication network. Since, however, the data transmitted in packets are asynchronous until entering the network, they will also be denoted below as asynchronous data for the sake of simplicity, as distinct from the source data, which are transmitted in a continuous data stream without a defined length and are also denoted as synchronous data.

According to a preferred embodiment, the subscriber address is transmitted with the start of the data packet with the asynchronous data. This permits simple, cost-effective and reliable decoding of the address, and thus reliable transmission of the asynchronous data from their source to their sink, which is determined by the subscriber address.

In the preferred embodiment of the invention, just like the bit positions for the asynchronous data, the reserved bit positions for the source data which are transmitted in a continuous data stream form in each case a contiguous region, the two regions for the synchronous and the asynchronous data adjoining one another. The two regions together preferably occupy a fixed number of bit positions within a bit group. The region for synchronous data can, in turn, be subdivided into a plurality of component bit groups, in order to form a plurality of transmission channels for synchronous data.

The boundary between the two regions can be set in accordance with the current network configuration, as described above. In accordance with an added feature of the invention, the boundary between the region for the source data (continuous data stream) and the region for the data (transmitted in data packets) is set in real-time (during ongoing operation) in accordance with the transmission capacities for synchronous or asynchronous data currently required. Synchronous data are preferably processed with priority in this case. This means that an ongoing audio operation, for example, is in no case interrupted when a data-intensive navigation message is to be transmitted by an 8× CD-ROM drive, but the transmission rate for the asynchronous data is reduced in accordance with the transmission capacity still available.

The available total transmission capacity can be utilized optimally at any point in time owing to the dynamic division of the region for source data in the bit groups. The size of the regions or the position of the boundary between the two regions for synchronous and asynchronous data can be specified in a data field which is formed by specific bit positions in each bit group. Each subscriber can easily determine the end of the synchronous data field and the start of the asynchronous data field from the information contained in that data field.

In accordance with an additional feature of the invention, 64 bytes are used for a complete bit group. Sixteen successive bit groups are combined in each case to form a block. As known in the pertinent art, in each bit group, within the regions reserved for the control data, it is possible to provide a data field as preamble, in particular for identifying the start of the bit group and, if appropriate, for identifying the start of a block or for identifying an assignment between partial bit groups (component bit groups) and specific subscribers.

Sixteen control bits which can be used to transmit control messages are provided within the regions reserved for the control data. The control bits of several successive bit groups (e.g. 16) are combined to form a control message.

Sixty bytes of a bit group are respectively provided for source data. Because of this large number of bytes, it is not necessary, as in the so-called SPDIF format (Sony/Philips digital-interface-format), which has become accepted as transmission format with the development of CD players, to provide separate bit groups for left-hand and right-hand audio channels. Instead, it is possible for the left-hand and right-hand audio channels to be respectively assigned to a transmission channel formed by a component bit group. Nevertheless, data can easily be converted from the SPDIF format into the format according to the invention, and vice versa.

In accordance with another feature of the invention, a specific bit position is allocated a parity identifier for the purpose of error detection in each bit group. The parity check is defined within the regions reserved for the control data. Furthermore, in each bit group, within the regions reserved for the control data, a data field comprising a plurality of bits (e.g. six bits) is provided which contains a numerical value corresponding to the position of a subscriber in the annular communication network. This numerical value enables each subscriber to detect its position in the ring easily, and this is useful for time-critical applications.

The coding on the individual bits is preferably performed by means of so-called biphase coding. In this way, the clock signal, encoded in the data signal, can be transmitted together with the latter inside the network. The clock pulse is preferably generated by any network subscriber which operates as a clock pulse generator, the remaining network subscribers operating synchronously with the clock pulse generator by being matched, for example, via PLL circuits to the received clock pulse.

A data packet which contains more bit positions than the region of a bit group which is reserved for the data which are transmitted in data packets can be transmitted in the regions, reserved for the data packets, of a plurality of successive bit groups. In order to indicate this to the receiver of the data packet, the novel method provides for a flag to be set which comprises one or more specific bit positions within the regions reserved for the control data. Each subscriber which finds this flag set in a bit group may not write into the regions of the next bit group which are reserved for asynchronous data. Each subscriber which is currently receiving detects with the aid of the set flag that transmission has not yet been terminated, and still does not confirm receipt.

In a communication network with a ring structure, each subscriber copies the data stream immediately to the next subscriber. During reception and transmission of the data signals by a subscriber, if appropriate with processing of the data in the subscriber, however, a short delay is produced which adds up to an appreciable delay after the traversal of a multiplicity of subscribers. In order to avoid data confusion or data losses on the basis of such time shifts, the clock pulse generator extracts the data to be transmitted from each bit group arriving at the clock pulse generator and copies them into the bit group following thereupon, with the result that a delay is produced in a complete traversal of the ring which corresponds precisely to the temporal length of a bit group. It is therefore necessary, for example, for control messages which are associated with the boundaries of blocks and bit groups, to make arrangements so that the control message is detected as such even after the clock pulse generator has been passed. A suitable method for this is

known from the European patent disclosure EP-A-0 725 519. The delay in the clock pulse generator is not a problem per se for the data packets in the transmission method according to the invention, which are not associated either with the boundaries of blocks or the boundaries of bit groups, and have an address. However, it must be ensured that the transmitter of a data packet extracts the latter actively from the data stream or rereleases the space occupied by the data packet after it has arrived at it again with a bit group delay when no further data packets are to be transmitted (if there are still data to be transmitted, it suffices when the transmitter overwrites the data sent so far). Otherwise, the occupied bit group would circulate forever in the network.

In accordance with again another feature of the invention, the occupation of the asynchronous region of a bit group with data is identified by providing a free/occupied identifier at the start of each asynchronous region. This identifier is set to "occupied" by the transmitter of a data packet while it is transmitting data, and reset to "free" after the transmitter has transmitted the last part of its data. Thereafter, the bit group is available again to all transmitters for asynchronous data in the network.

An alternative possibility of preventing an occupied bit group from circulating forever in the network consists in a specific subscriber, preferably the clock pulse generator, providing each data packet, which passes it or which it transmits itself, with a flag. The clock pulse generator deletes the data packet or cancels a corresponding occupied identifier when it detects two data packets with the same flag.

The invention provides a data transmission method for synchronous and for asynchronous data which is compatible with conventional synchronous systems, without sacrificing in transmission capacity. As a result, the transmission rate itself can be increased even when the clock frequency can no longer be increased without generating radiated interference. The latter, of course, can be suppressed only with very considerable outlay.

At the same time, the data transmission method according to the invention is particularly economic. Although buffers are required for feeding asynchronous data into the network, relatively few buffer memories are required because of the ability of the data transmission format according to the invention to adapt to the current conditions in the system.

For the above reasons, the method according to the invention is advantageously used, in particular, in stationary domestic communication systems and in mobile communication systems in motor vehicles.

In accordance with a concomitant feature of the invention, the network is wired with optical fibers between the individual subscribers which permit high data transmission rates. In the case of a communication system in a motor vehicle, moreover, the low weight of optical fibers is particularly advantageous. However, the invention is also suitable for purely electric communication ring-networks in which the line segments are coaxial cables, for example.

Other features which are considered as characteristic for the invention are set forth in the appended claims.

Although the invention is illustrated and described herein as embodied in a method for transmitting source data and control data in a communication system with a ring structure, it is nevertheless not intended to be limited to the details shown, since various modifications and structural changes may be made therein without departing from the spirit of the invention and within the scope and range of equivalents of the claims.

The construction and method of operation of the invention, however, together with additional objects and advantages thereof will be best understood from the following description of specific embodiments when read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a timing diagram of the data transmission format, according to the invention, in a ring-shaped communication network;

FIG. 2 is a schematic view of a ring-shaped network with four network subscribers;

FIG. 3 is a timing diagram representing the transmission of a data packet from one network subscriber to another network subscriber of the network of FIG. 2;

FIG. 4 is a diagram of a data packet according to FIG. 3 as it is transmitted and received, respectively, by the various network subscribers of FIG. 2;

FIG. 5 is a timing diagram representing the transmission of a data packet from one network subscriber to another network subscriber of the network of FIG. 2, the transmission of the data packet being performed via the clock pulse generator;

FIG. 6 is a diagram of a data packet according to FIG. 5 as it is transmitted and received, respectively, by the network subscribers of FIG. 2; and

FIG. 7 is a diagrammatic view illustrating a division of a long data packet into successive bit groups.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to the figures of the drawing in detail and first, particularly, to FIG. 1 thereof, there is seen a bit group 1, also termed a frame. The bit group 1 comprises 64 bytes, which is 512 bits. This is an even multiple of the lengths of bit groups in known transmission formats for synchronous data, in particular SPDIF format or the format known from the above-mentioned EP-A-0 725 522. Full compatibility is ensures with these formats because of the simple convertibility.

The bit group 1 contains a preamble 2 which comprises four bit positions 3. The preamble 2 permits the PLL circuit of a subscriber to lock on to a received clock pulse. Sixteen bit groups 1 are combined in each case to form a block, the first preamble 2 of each block containing a special block identifier bit.

A data field 4 adjoins the preamble 2 which comprises four bit positions 5 and whose function will be explained further below.

A region 6 for synchronous data adjoins the data field 4. The region 6 can be subdivided in a known way into a plurality of component bit groups 7 of identical length which are assigned in each case to a specific subscriber. The assignment between the component bit groups 7 and the respective subscribers is fixed in the preamble 2.

A region 8 for asynchronous data adjoins the region 6 for synchronous data. The region 6 for synchronous data and the region 8 for asynchronous data together occupy 60 bytes of the bit group 1. The total transmission capacity of the network for synchronous and asynchronous source data is formed by these 60 bytes. The length of the region 8 for asynchronous data can be 0, 4, 8, . . . 56 or 60 bytes, and is fixed by a value ASY (ASY=0, 1, 2, . . . 15) which is stored in the region 4 preceding the region 6 for synchronous data.

The region 6 for synchronous data thus comprises 60-(4× ASY) bytes. The region 8 for asynchronous data is followed by a data field 9, with a size of 16 bits (2 bytes), for control bits. The control bits of a block, i.e., 32 bytes or 192 bits, form a control message.

A data field 10 with 6 bits adjoins the data field 9 for control bits. A subscriber generating clock pulses, or a clock pulse generator, writes a "0" into the data field 10, and the value in the data field 10 is increased by one by each following subscriber in the ring, with the result that each subscriber—in this case a maximum of 64 subscribers—has direct knowledge of its position in the ring.

The data field 10 is followed by a single bit position 11 which indicates whether a data packet in the region 8 for asynchronous data is continued in a bit group following the bit group 1 or not. The bit position 11 is followed by a bit position 12 for a parity bit for error detection.

The regions 6 and 8 of each bit group 1 are reserved for synchronous or asynchronous source data, and the data fields 2, 4, 9, 10 and two individual bit positions 11 and 12 are reserved for synchronous control data.

It will be appreciated that the division of the bit group 1 is not illustrated true to scale for the purpose of ensuring a detailed representation of individual bit positions.

A data packet 13 which is to be transmitted over the network by a subscriber operating in a packet-orientated fashion is inserted into the region 8 for asynchronous data in such a way that it starts at the boundary 14 between the region 6 for synchronous data and the region 8 for asynchronous data, as represented by means of dashed lines. The data packet 13 contains a header region 15 with the address of a receiver, connected to the network, of the data packet 13. If the transmitting subscriber is a device which outputs data packets without addresses, an address is to be added to each of these data packets. This can easily be carried out in an interface between the subscriber and the network.

In the example represented in FIG. 1, the data packet 13 is somewhat shorter than the region 8 for asynchronous data, so that it can be transmitted in one single bit group 1. Where a data packet is longer than the region 8 for asynchronous data in one bit group 1, the rest of the data packet is transmitted in the bit groups following the bit group 1 and, if necessary, in further bit groups. This division is indicated by the flag at the bit position 11. The transmission of a data packet which is longer than the region 8 for asynchronous data in a bit group 1 is explained in detail further below with reference to FIGS. 2 to 7.

The dynamic management of the boundary 14 between the region 6 for synchronous data and the region 8 for asynchronous data is explained with the aid of the following example.

In an annular communication network in a motor vehicle there are interconnected via optical fibers: a radio receiver, a CD player, a mobile telephone, an input/output unit for speech, a navigation system, which comprises a GPS receiver and an 8× CD-ROM drive as database for map material, a plurality of amplifier/loudspeaker combinations, and a viewing screen.

It may be assumed that initially only the radio receiver and the amplifier/loudspeaker combinations are active in the communication network, and that initially 60 bytes are reserved for transmitting synchronous source data. In other words, the region 8 for asynchronous data comprises zero bytes. The audio data from the radio receiver are transmitted via some of the channels formed by the component bit group 7 to the amplifier/loudspeaker combinations, the majority of the 60 bytes of transmission capacity remaining free.

If the navigation system, for example, is now activated, the region 6 for synchronous data is automatically reduced to such an extent that just enough transmission capacity remains for the audio data and for video data from the navigation system to the viewing screen. The region 8 for asynchronous data is correspondingly enlarged so that a comparatively great capacity is available for the data-intensive packet-oriented communication between the navigation system and the CD-ROM drive. Synchronous source data enjoy priority over asynchronous data, however. That is to say, if a call has in the meantime arrived at a telephone with synchronous operation, the region 6 for synchronous data is automatically enlarged so as to accommodate the added synchronous data transfer.

FIG. 2 shows a ring-shaped or annular network with a subscriber 20 used as clock pulse generator and three further subscribers 21, 22 and 23. The four subscribers 20, 21, 22 and 23 are interconnected annularly via optical-fiber segments 24, 25, 26 and 27. The physical direction of the data transmission is represented by arrows on the optical-fiber segments.

FIG. 3 represents a transmission of asynchronous data from the subscriber 21 to the subscriber 23. The transmission, therefore, does not traverse the clock pulse generator 20. The clock pulse generator 20 transmits bit groups at specific time intervals, four regions 30, 31, 32 and 33 for asynchronous data from four successive bit groups being illustrated in FIG. 3. The region 30 transmitted by the clock pulse generator 20 is empty, and this is indicated by a "free" identifier immediately at the start of the region 30. The subscriber 21 detects that the region 30 is free, and checks its own transmit status. In the case of a transmit request, the subscriber 21 identifies the region 30 as occupied (identifier "occ.") and immediately begins to send an address D0 and data D1 and D2 via the region 30. In the example shown here, the dataset to be transmitted is greater than the number of free bytes which is contained in a region for asynchronous data of a bit group, for which reason the subscriber 21 sets a bit 34 to "1". The bit 34 corresponds to the bit position 11 in the bit group format of FIG. 1, but is illustrated here immediately adjacent the region 30 for reasons of clarity. The bit 34 set signals to all further subscribers that the data packet is continued in the next bit group or the region thereof for asynchronous data. The downstream subscribers then simply do not attempt to write into this region or are not allocated this region if the allocation is performed from a control center. Furthermore, the set bit 34 signals to the receiver of the data packet—here the subscriber 23—that it must remain ready to receive beyond the bit group boundary.

The subscriber 22 detects the occupied region 30, checks the address D0, detects that there is no correspondence and behaves in a transparent fashion. The transmission is delayed for only a brief moment.

The subscriber 23 detects its address D0 in the region 30 and begins to receive.

The clock pulse generator 20 receives the bit group, generated by it and now occupied in the region 30 with the transmission data from the subscriber 21, with a delay dependent on the size of the network. The contents of the received region 30 (together with the remaining contents of the bit group in which the region 30 is contained) are copied into the region 31 for asynchronous data of the next bit group, as indicated by hatched arrows. For this purpose, the clock pulse generator 20 has an intermediate memory of appropriate size.

The subscriber 21 then overwrites the data D0, D1 and D2 with successor data D3, D4 and D5. Assuming that this is the last part of the data to be sent, the subscriber 21 resets the bit 34 to "0", in order to signal that the region for asynchronous data of the next bit group is available again for transmission to other subscribers.

The bit group is then transmitted from the subscriber 22 without change in the region 31 to the receiving subscriber 23, and from there to the clock pulse generator 20, which copies the contents of the region 31 into the region 32 of the next bit group generated by it. When the subscriber 21 receives this bit group, it resets the "occupied" identifier to "free". This prevents this bit group from always circulating in the system as being occupied with asynchronous data if the clock pulse generator 20, as in this example, basically copies the contents of the last received bit group into the next bit group, without evaluating the contents thereof. The transmission of the complete data packet takes a time t.

FIG. 4 shows the regions 30, 31, 32 and 33 for asynchronous data in the successive bit groups, as they are seen by the individual subscribers in the transmission of FIG. 3.

FIG. 5 represents the case in which the transmission link leads from the transmitter, in this case the subscriber 23, to the receiver, which is the subscriber 21 in this case, via the clock pulse generator 20. The clock pulse generator 20 again generates a bit stream which contains no asynchronous data. The subscribers 21 and 22 detect the free region 30 and check their own respective transmit status. Since there is no transmit request in the case of the two subscribers 21 and 22, they relay the bit group with the region 30 in a transparent fashion.

The subscriber 23 would like to transmit, however, for which reason it sets the "occ." identifier (=occupied) and begins to transmit data immediately. Since the dataset to be transmitted is larger than the number of free bits which is contained in a region for asynchronous data of a bit group, the bit 34 is set to "1".

The clock pulse generator 20 receives the bit group generated by it and now occupied in the region 30 with the transmission data from the subscriber 21, with a delay dependent on the size of the network. The contents of the received region 30 of the re-received bit group are copied, as in the example of FIG. 3, into the region 31 for asynchronous data of the next bit group generated.

The subscriber 21 detects the occupied region 31 delayed by a bit group with reference to the clock pulse generator 20, detects its address D0 and begins receiving.

When the subscriber 23 transmits the last part of its data, it resets the bit 34 to "0", in order to signal that the region for asynchronous data of the next bit group is available again for transmission to other subscribers.

The subscriber 23 receives the data transmitted by it in the region 32 of the next bit group, into which they have been copied from the clock pulse generator 20, and resets its "occupied" identifier to "free". Just as in the example of FIG. 3, the bit group with the free region is detected with a delay of one bit group.

FIG. 6 shows the regions 30, 31, 32 and 33 for asynchronous data in the successive bit groups, as they are seen by the individual subscribers in the case of the transmission of FIG. 5.

In the cases in which the data are not transmitted via the clock generator 20 (FIGS. 3 and 4), a subscriber detects a message intended for it in the same bit group into which it is inserted by a transmitting subscriber, and in the cases in which the data are transmitted via the clock pulse generator 20 (FIGS. 5 and 6), a subscriber detects a message intended for it in bit groups following thereupon.

FIG. 7 shows a data packet 35 which contains a free/occupied identifier 36, an address 37 and data 38, and which is longer than the region 8 for asynchronous data of a bit group, which corresponds to the bit group 1 of FIG. 1. The data packet 35 is divided into portions A, B, . . . whose length corresponds at most to the length of a region 8 for asynchronous data. The individual portions are inserted into one bit group in each case, as is shown diagrammatically. In addition, the bit position 11 is set to "1" within each bit group so long as the data packet 35 has not yet been accommodated completely in bit groups. Upon complete transmission of the entire packet 35 (in portions A, B, . . .) the bit position 11 is set back to "0".

We claim:

1. A method for the common transmission of digital source data and control data between data sources and data sinks, which are subscribers to a communication network with a ring structure, the method which comprises:

transmitting source data and control data in a continuous data stream synchronized to a clock signal and in a format which prescribes a pulsed sequence of individual bit groups of equal length in which specific bit positions are reserved for source data and control data;

defining data packets each having a start and a defined length, and carrying a subscriber address;

reserving an arbitrarily large contiguous region of the bit positions for the source data within a bit group and transmitting the data packets within the contiguous region.

2. The method according to claim 1, wherein the defining step comprises assigning the subscriber address to the start of a respective data packet.

3. The method according to claim 1, which comprises writing, with a subscriber sending data packets, successive bits of a data packet to be transmitted in adjacent bit positions within the contiguous region.

4. The method according to claim 3, wherein the defining step comprises defining a data packet with more bit positions than the contiguous region of a given bit group and wherein the writing step comprises writing the data packet into a plurality of successive bit groups.

5. The method according to claim 4, which comprises setting a flag in a bit group in which data of a data packet are transmitted which is continued in a subsequent bit group, the flag indicating that the data packet is continued in the subsequent bit group.

6. The method according to claim 5, which further comprises defining regions for the control data in each bit group, and reserving one or more specific bit positions for the flag in each bit group within the regions for the control data.

7. The method according to claim 1, which comprises reserving a contiguous region of bit positions for the source data which are transmitted in a continuous data stream, and wherein the region for the source data which are transmitted in a continuous data stream and the region for the data which are transmitted in data packets adjoin one another.

8. The method according to claim 7, wherein the region for the source data which are transmitted in a continuous data stream and the region for the data which are transmitted in data packets are formed together by a fixed number of bit positions within a bit group.

9. The method according to claim 8, which further comprises permanently setting a boundary between the region for the source data which are transmitted in a continuous

data stream and the region for the data which are transmitted in data packets.

10. The method according to claim 8, which further comprises setting a boundary between the region for the source data which are transmitted in a continuous data stream and the region for the data which are transmitted in data packets in real-time operation in accordance with a currently required transmission capacity.

11. The method according to claim 10, wherein the setting step comprises giving priority to source data which are transmitted in a continuous data stream.

12. The method according to claim 7, which comprises providing each bit group with a data field for information relating to a size of the region for source data in the continuous data stream or the size of the region for the data in the data packets.

13. The method according to claim 7, which further comprises subdividing the region within a bit group for the source data which are transmitted in a continuous data stream into a plurality of component bit groups of equal length, and assigning the source data allocated to each component bit group to a specific subscriber as a function of the control data.

14. The method according to claim 1, which further comprises sending a data packet with a given subscriber, and deleting or overwriting the data packet with new data by the given subscriber after the data packet has traversed the ring-shaped network and arrived at the given subscriber.

15. The method according to claim 1, wherein each bit group is defined with a region for asynchronous data transfer, and wherein the region for asynchronous data transfer includes a data field at a start of the region indicating whether the given bit group is occupied with asynchronous data or free.

16. The method according to claim 15, which further comprises writing, with a specific subscriber, a flag into the data packets, and deleting an identifier of a bit group as being occupied with asynchronous data when two data packets with the same flag are detected.

17. The method according to claim 16, wherein a subscriber providing a clock pulse is defined as the specific subscriber.

18. The method according to claim 1, wherein a complete bit group is defined with 64 bytes.

19. The method according to claim 1, which further comprises forming a block by combining sixteen successive bit groups.

20. The method according to claim 1, which comprises defining a preamble in each bit group within a region reserved for the control data, the preamble being a data field identifying a start of the bit group, a start of a block of bit groups, or an assignment between partial bit groups and specific subscribers.

21. The method according to claim 1, which comprises defining sixteen control bits within each bit group in the region reserved for the control data, and combining the control bits of sixteen successive bit groups to form a control message.

22. The method according to claim 1, which further comprises defining a parity identifier in a bit position in each bit group within the region reserved for the control data.

23. The method according to claim 1, which further comprises defining in each bit group, within a region reserved for the control data, a data field with a plurality of bits, the data field containing a numerical value corresponding to a position of a respective subscriber in the ring-shaped communication network.

24. The method according to claim 1, wherein the subscribers are connected via optical fibers and the transmitting step comprises transmitting through the optical fibers.

25. The method according to claim 1, wherein the communication network is a stationary communication system and the transmitting step comprises transmitting data among subscribers in the stationary communication system.

26. The method according to claim 1, wherein the communication network is a mobile communication system and the transmitting step comprises transmitting data among subscribers in the mobile communication system.

27. The method according to claim 26, wherein the mobile communication system is a communication system in a motor vehicle.

* * * * *

US006108346A

United States Patent [19]

Doucette et al.

[11] Patent Number: 6,108,346

[45] Date of Patent: Aug. 22, 2000

[54] **COMBINED SYNCHRONOUS AND ASYNCHRONOUS MESSAGE TRANSMISSION**

[75] Inventors: **John Doucette**, Londonderry; **Thomas J. Bryden**, Peterborough; **Todd Byron**, Manchester, all of N.H.

[73] Assignee: **Xlox Corporation**, Burlingame, Calif.

[21] Appl. No.: **09/268,099**

[22] Filed: **Mar. 13, 1999**

Related U.S. Application Data

[60] Provisional application No. 60/098,297, Aug. 27, 1998.

[51] **Int. Cl.**[7] **H04L 12/403**

[52] **U.S. Cl.** **370/450**; 370/353; 370/468

[58] **Field of Search** 370/352, 353, 370/354, 452, 460, 450, 468, 909, 470, 471, 472, 473, 476; 709/251, 238

[56] **References Cited**

U.S. PATENT DOCUMENTS

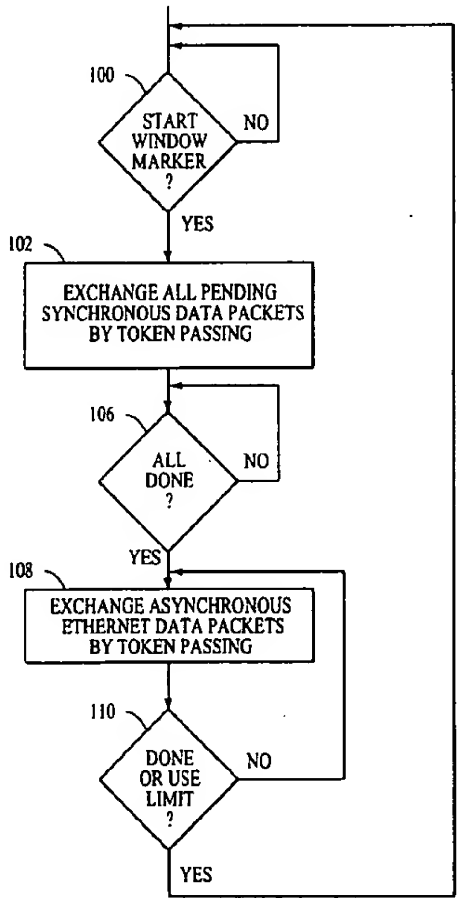
5,392,280	2/1995	Zheng	370/60
5,570,355	10/1996	Dial et al.	370/352
5,935,214	9/1999	Stiegler et al.	709/251

Primary Examiner—Huy D. Vu
Assistant Examiner—Kevin C. Harper
Attorney, Agent, or Firm—Elmer Galbi

[57] **ABSTRACT**

A communication system including a collection of modules coupled in a ring architecture which integrates synchronous and asynchronous message transmission. Asynchronous data packets and synchronous voice packets are exchange on a single communication link. Packetized information exchange references a fixed length window with all synchronous data being exchanged at the beginning of each window and with asynchronous data exchanged during the remaining portion of each window. Virtual circuits utilizing the synchronous packets can deliver telephone conversations without degradation in voice quality and yet the system can also transmit asynchronous data packets.

2 Claims, 5 Drawing Sheets



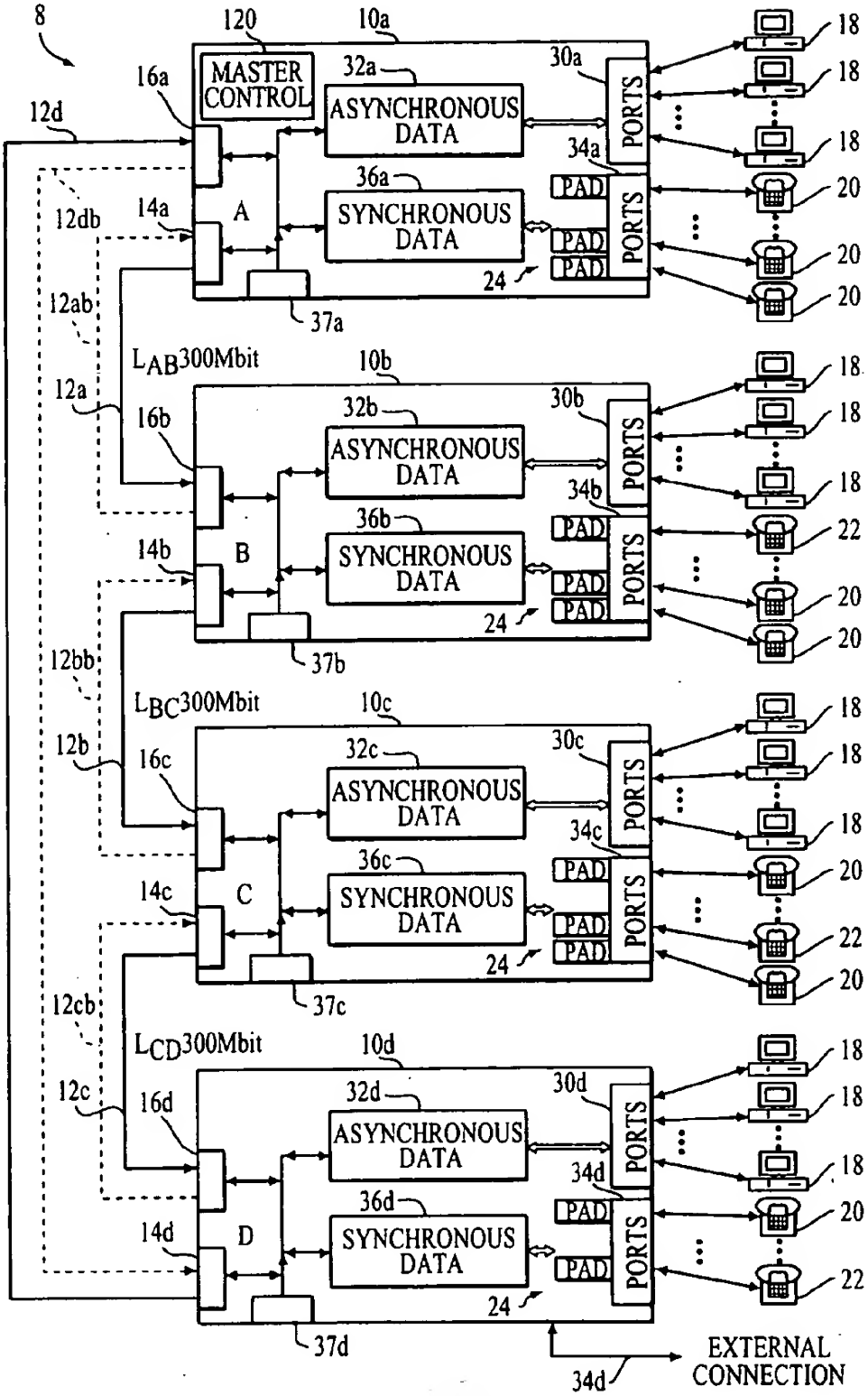

8
120
10a
MASTER CONTROL
16a
12d
32a
ASYNCHRONOUS DATA
30a
PORTS
34a
PAD
36a
SYNCHRONOUS DATA
12db
14a
A
24
37a
12ab
18
20
22
L_{AB}300Mbit
10b
12a
16b
32b
30b
34b
36b
14b
B
37b
12bb
L_{BC}300Mbit
10c
12b
16c
32c
34c
36c
14c
C
37c
12cb
L_{CD}300Mbit
10d
12c
16d
32d
34d
36d
14d
D
37d
EXTERNAL CONNECTION

FIG. 1

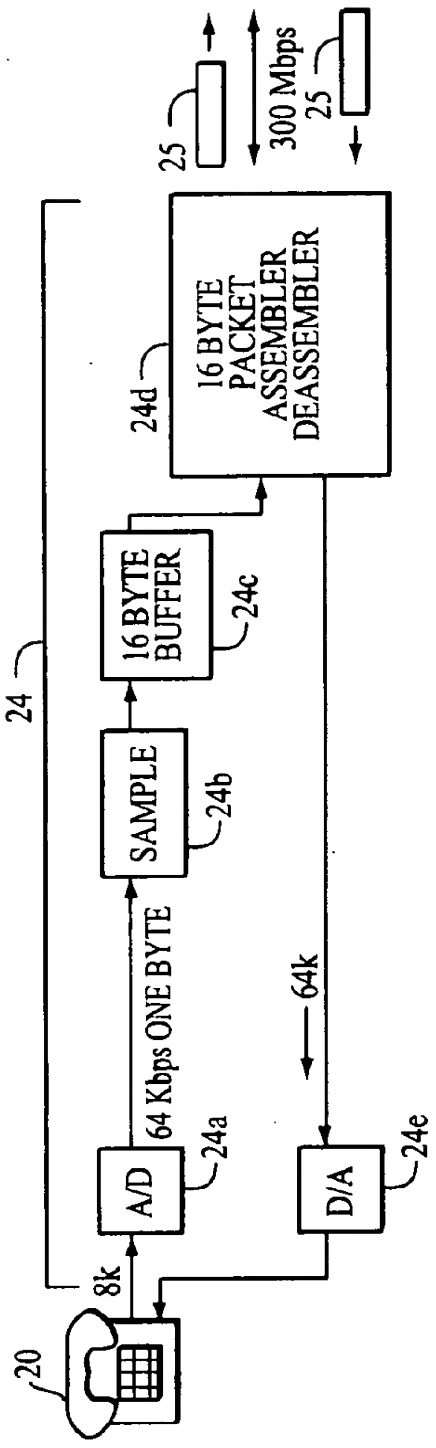

25
300 Mbps
25
16 BYTE PACKET ASSEMBLER DEASSEMBLER
24d
16 BYTE BUFFER
24c
SAMPLE
24b
24
64 Kbps ONE BYTE
64k
A/D
24a
D/A
24e
8k
20

FIG. 2

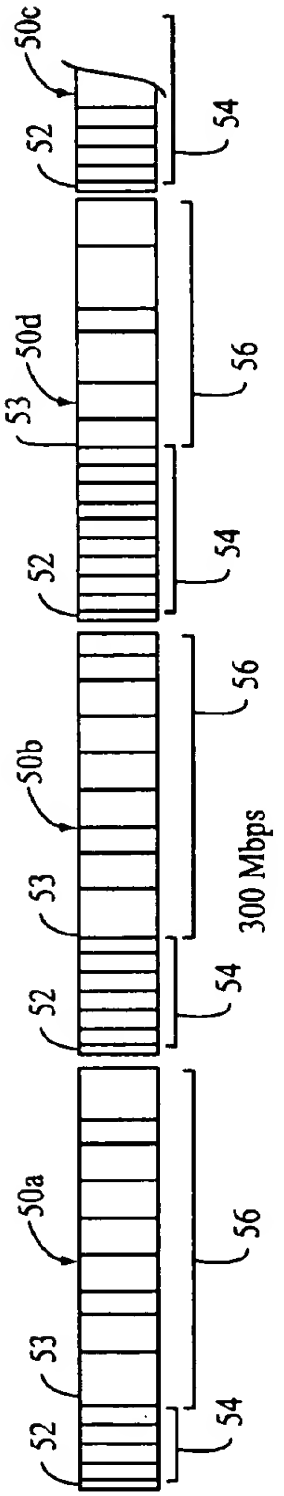

50c
52
54
50d
53
56
52
54
50b
53
56
300 Mbps
52
54
50a
53
56
52
54

FIG. 3

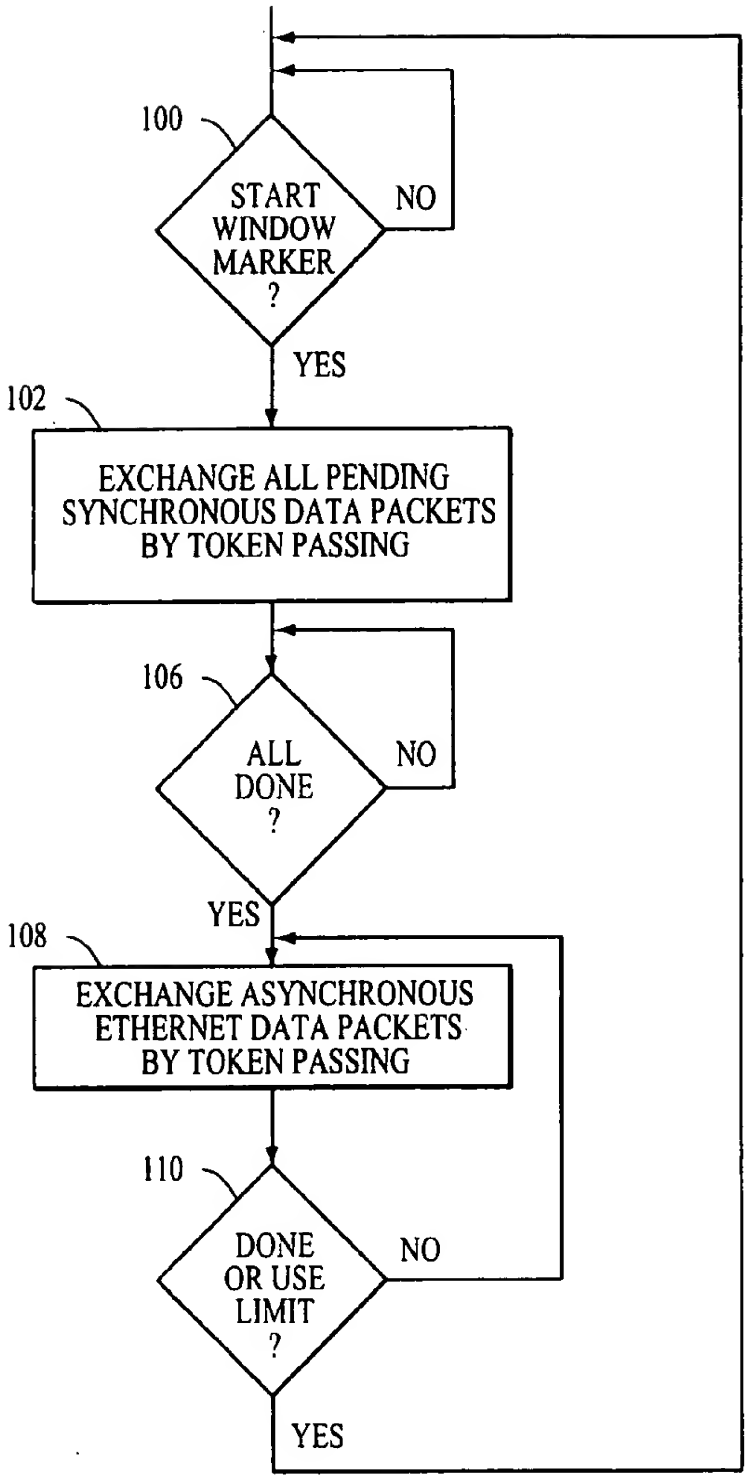

100
START WINDOW MARKER ?
NO
YES
102
EXCHANGE ALL PENDING SYNCHRONOUS DATA PACKETS BY TOKEN PASSING
106
ALL DONE ?
NO
YES
108
EXCHANGE ASYNCHRONOUS ETHERNET DATA PACKETS BY TOKEN PASSING
110
DONE OR USE LIMIT ?
NO
YES

FIG. 4

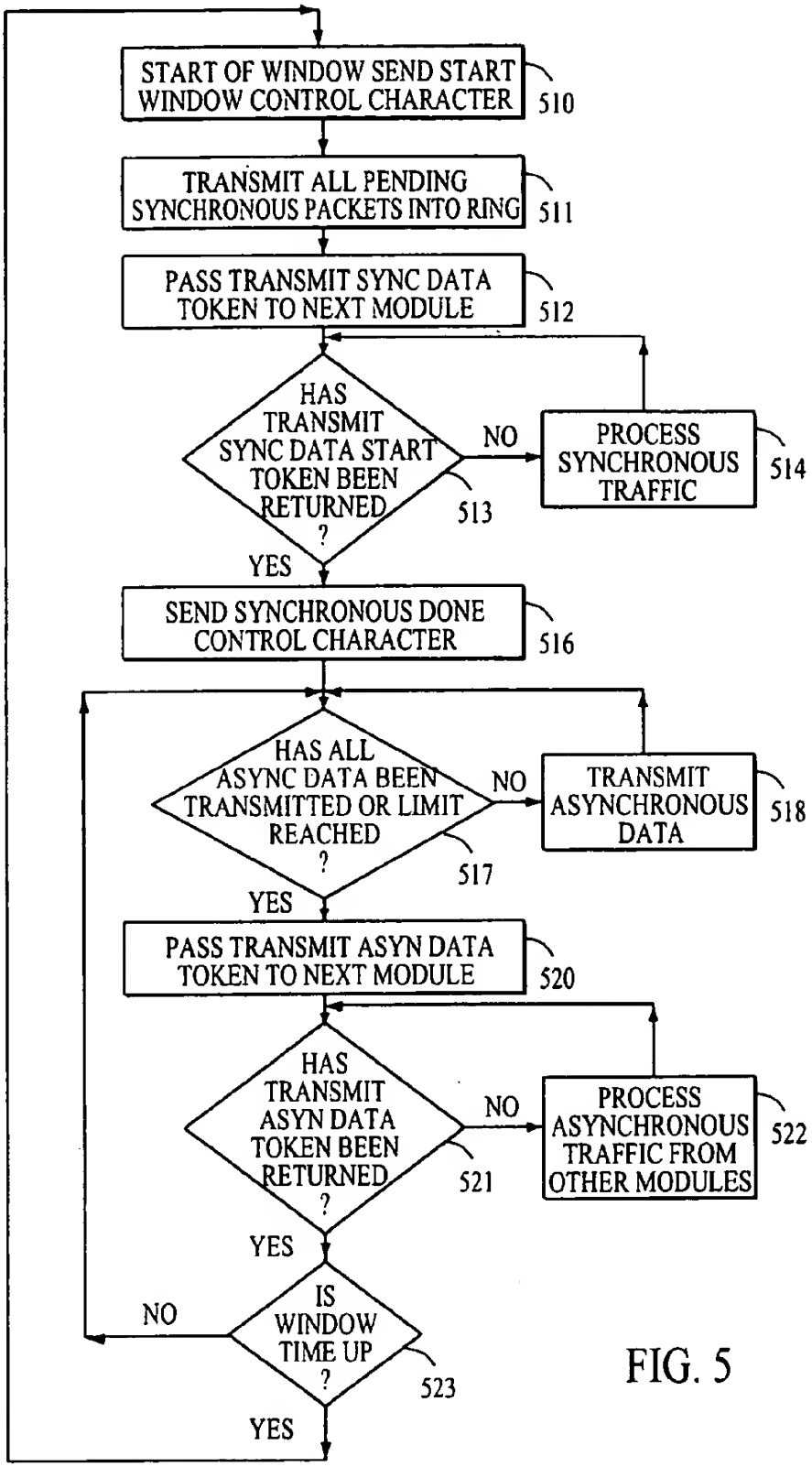
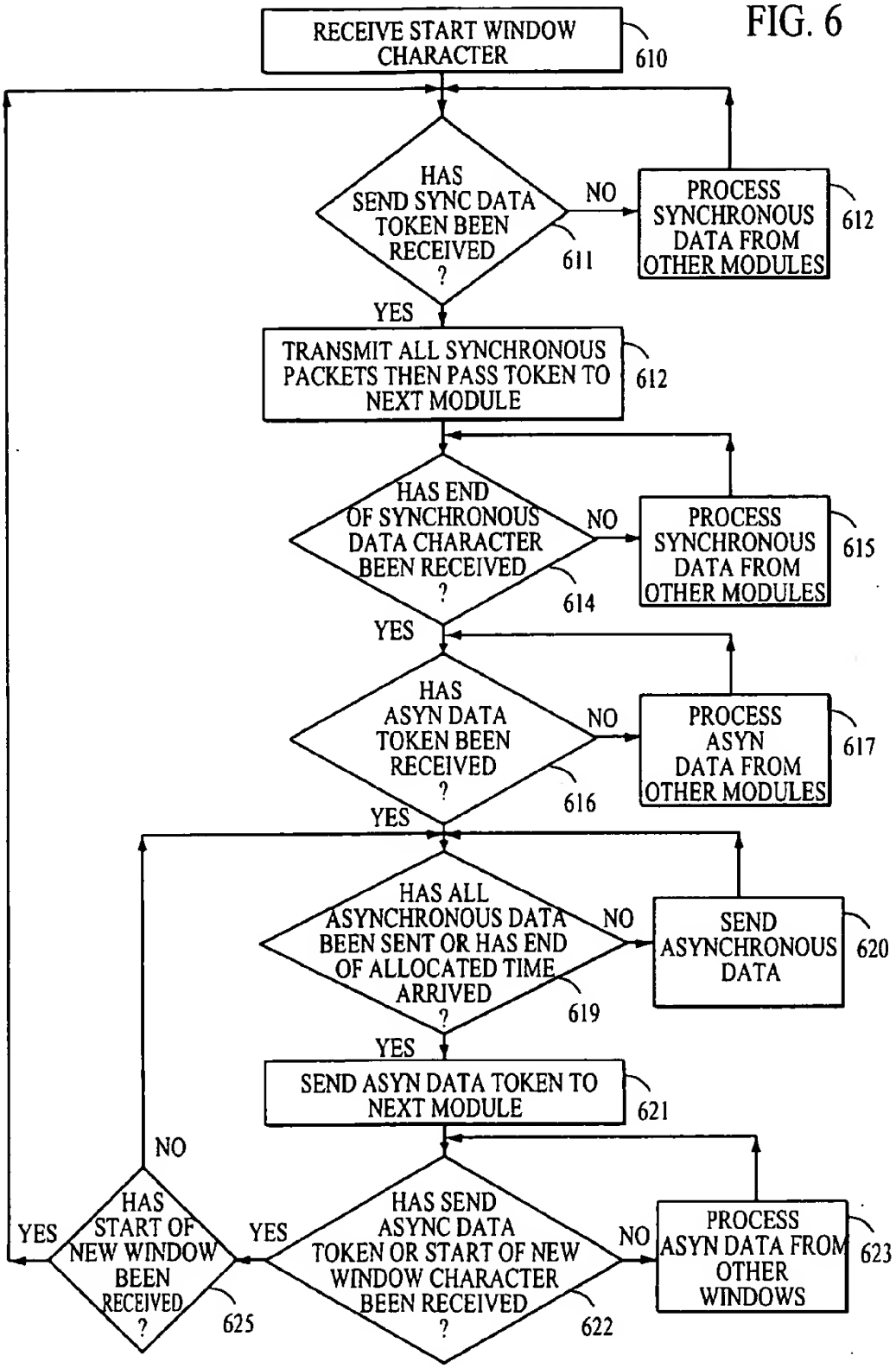

START OF WINDOW SEND START WINDOW CONTROL CHARACTER
510
TRANSMIT ALL PENDING SYNCHRONOUS PACKETS INTO RING
511
PASS TRANSMIT SYNC DATA TOKEN TO NEXT MODULE
512
HAS TRANSMIT SYNC DATA START TOKEN BEEN RETURNED ?
513
NO
PROCESS SYNCHRONOUS TRAFFIC
514
YES
SEND SYNCHRONOUS DONE CONTROL CHARACTER
516
HAS ALL ASYNC DATA BEEN TRANSMITTED OR LIMIT REACHED ?
517
NO
TRANSMIT ASYNCHRONOUS DATA
518
YES
PASS TRANSMIT ASYN DATA TOKEN TO NEXT MODULE
520
HAS TRANSMIT ASYN DATA TOKEN BEEN RETURNED ?
521
NO
PROCESS ASYNCHRONOUS TRAFFIC FROM OTHER MODULES
522
YES
IS WINDOW TIME UP ?
523
NO
YES

FIG. 5

FIG. 6


RECEIVE START WINDOW CHARACTER
610
HAS SEND SYNC DATA TOKEN BEEN RECEIVED ?
611
NO
PROCESS SYNCHRONOUS DATA FROM OTHER MODULES
612
YES
TRANSMIT ALL SYNCHRONOUS PACKETS THEN PASS TOKEN TO NEXT MODULE
612
HAS END OF SYNCHRONOUS DATA CHARACTER BEEN RECEIVED ?
614
NO
PROCESS SYNCHRONOUS DATA FROM OTHER MODULES
615
YES
HAS ASYN DATA TOKEN BEEN RECEIVED ?
616
NO
PROCESS ASYN DATA FROM OTHER MODULES
617
YES
HAS ALL ASYNCHRONOUS DATA BEEN SENT OR HAS END OF ALLOCATED TIME ARRIVED ?
619
NO
SEND ASYNCHRONOUS DATA
620
YES
SEND ASYN DATA TOKEN TO NEXT MODULE
621
NO
HAS START OF NEW WINDOW BEEN RECEIVED ?
625
YES
HAS SEND ASYNC DATA TOKEN OR START OF NEW WINDOW CHARACTER BEEN RECEIVED ?
622
YES
NO
PROCESS ASYN DATA FROM OTHER WINDOWS
623

COMBINED SYNCHRONOUS AND ASYNCHRONOUS MESSAGE TRANSMISSION

RELATED APPLICATION

The present invention is a continuation-in-part of application Ser. No. 60/098,297 filed Aug. 27, 1998 entitled "Combined Synchronous and Asynchronous Message Transmission."

BACKGROUND OF THE INVENTION

Packet switching systems transmit data by breaking the data into relatively small manageable pieces called packets. Packet switching can be used to transmit data in both computer networks and in telephone voice networks. Telephone packet switching networks transmit a series of packets over the same route in the network. Such systems in effect establish a virtual circuit from the point where a series of packets enters the network to the point where the packets are delivered. Packet switching networks establish virtual circuits through the network in order to transmit voice without delay and distortion.

Protocols such as the Internet ITPC protocol can transmit voice without establishing a virtual circuit connection, however, voice transmission using this type of protocol generally has less quality than voice transmitted using protocols which establish virtual circuits between the input point and the output point in the network.

Today, some voice transmissions are being made over packet protocols (such as the Internet) which do not establish virtual circuits. Voice connections over such circuits are of relatively low quality. The packet protocols which are used in the public telephone network are packet protocols which establish virtual circuits and which transmit all the packets that constitute a conversation over the same route through the network. Thus they provide high quality connections.

Data communication protocols can be characterized as either synchronous or asynchronous. Examples of widely used synchronous protocols are the X.25 protocol, and the frame relay protocol. Examples of widely used asynchronous protocols are the Ethernet, FDDI and ATM protocols. The X.25 protocol, the frame relay protocol and the ATM protocol are widely used in telephone systems. The Ethernet protocol and the FDDI ring protocol are widely used in local area networks (LANS) and wide area networks (WANS) that are used to interconnect computer systems.

There are various well known techniques for controlling asynchronous networks. One technique termed "carrier sense, multiple access with collision detection (CSMA/CD) is used in Ethernet networks. Another technique called token passing is used in FDDI ring networks.

Explanations of various synchronous and asynchronous protocols, and an explanation of CSMA/CD and FDDI ring networks is for example given in a book entitled "Voice and Data Communications Handbook" by Regis J. Bates and Donald Gregory which is published by McGraw Hill.

SUMMARY OF THE INVENTION

The present invention provides a ring protocol and system that combines synchronous and asynchronous transmission techniques. The ring can interconnect a number of modules and be utilized to transmit both fixed and variable packets between the modules. Communication time is broken into a sequence of fixed length windows. At the beginning of each window the modules communicate using a synchronous protocol. That is, at the beginning of each window, if any unit has synchronous traffic, such traffic is transmitted using a synchronous ring protocol and fixed length packets. Virtual circuits can be established between the modules using the synchronous fixed length packets communicated at the beginning of each window. When it is desired to establish a virtual circuit between any of the modules in the ring, each module is assured that at the beginning of each window, space will be allocated to transmit a synchronous fixed length packet to another module in the ring. The windows occur frequently enough that a virtual voice grade circuit can be established between the modules. After all synchronous packets required during any window have been transmitted, asynchronous variable length data packets are transmitted around the ring. Limits are provided relative to the number of asynchronous packets any one module can transmit, thereby avoiding monopolization of the ring by any one module. The modules are synchronized by a periodically circulating a timing control character around the ring.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is an overall block diagram of the preferred embodiment.

FIG. **2** is a diagram of the package assembler and disassembler (PAD) portion of the modules for coupling to analog telephone devices.

FIG. **3** shows a repeating sequence of fixed-length time windows used in allocating data traffic between the modules of FIG. **1**.

FIG. **4** is a system-level flow chart illustrating overall operation of the communication system of FIG. **1**.

FIG. **5** is a program flow chart showing the operation of the module which executes certain control among the modules of FIG. **1**.

FIG. **6** is a flow chart showing the operation of modules of FIG. **1** when receiving and processing information exchanged there-between.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A preferred embodiment of the invention is shown in FIG. **1** and described herein. FIG. **1** shows a communication system **8** which supports integrated exchange of asynchronous and synchronous data between a number of modules **10***a*, **10***b*, etc. The data exchanged between the modules includes data traffic from and to computers **18** and voice traffic from conventional telephone devices **20** and **22**. The data traffic from and to the computers **18** can include all of the various types of data traffic conventionally generated by computers such as video data, pc-phone data, etc. The system shown in FIG. **1** establishes "virtual circuits" between modules **10** for telephone traffic (i.e. synchronous transfer) and to also manages exchange of asynchronous information transfer between modules **10**.

As shown in FIG. **1** system **8** includes a collection of modules **10** organized in a ring architecture. Information travels from one module **10** to a successive module **10** as required. The information on the ring can be divided into three categories, namely, data packets, tokens, and control characters (including a timing character). As used herein the following terms have the following meaning.

A "byte" consists of ten bits. Eight data bits are coded into a ten bit byte. Since a ten bit byte is used to encode 8 bits of data, a byte can be decoded into 256 different data words

plus 768 additional decodes. Some of the additional decodes are used to form control characters, a timing character, and tokens. Other ones of the additional decodes are used for purposes unrelated to the present invention. Such decoding is conventional.

A "control character" consists one byte. The specific byte that forms each control character is selected from the decodes which do not form data words. There are three control characters which are used to implement the present invention. One control character indicates the start of a window, a second indicates the end of synchronous data transfer, and the third indicates the end of a window. When a module receives a control character it immediately retransmits the character to the next module in the ring.

A "token" consists of two bytes of data. As with control characters, the specific bytes that form each token are selected from the decodes which are not otherwise assigned. When a module receives a token, it only retransmits the token to the next module if certain conditions have been met. There are two tokens used in the implementation of the present invention. One token indicates that a module should begin transmitting its synchronous data and the second indicates that a module should begin transmitting asynchronous data. A module only passes a token to the next module after a module that receives a token has completed the task initiated by the token.

A "timing character" consists of one byte. This one byte is selected from the decodes not otherwise assigned.

A "synchronous packet" consists of 16 bytes of data plus three bytes of address information.

A "asynchronous packet" consists of 64 to 1524 bytes of data plus an 8 byte Ethernet header.

As each module **10** receives bytes (i.e. information packets, control characters, and tokens) from its predecessor in the ring, the module copies the bytes it receives internally and retransmits the bytes to a successive module **10** if appropriate. Bytes thereby flow at high speed around the ring architecture from one module **10** to another module **10**. It is noted that tokens are only transmitted from one module to another module when a task initiated by the token has been completed.

The ring architecture allows use of a variable number of modules **10**. Four such modules **10**, individually **10***a*–**10***d*, are shown in the particular embodiment described herein. It will be understood, however, that more modules **10** may be inserted into the ring architecture or that some of modules **10***a*–**10***d* may be removed from the ring architecture. Thus, modules **10** "stack" to meet use requirements, e.g., system **8** expands, to follow a growing user population or capacity requirement.

Each module **10** communicates with two adjacent modules through interconnecting communication links **12**. Each of the links **12** is bi-directional. For example link **12***a* goes from module **10***a* to **10***b* and link **12***ab* goes from module **10***b* to module **10***a*. In normal operation the system uses links **12***a*, **12***b*,**12***c*, and **12***d*. If one of these links is down (i.e. broken) the system automatically switches to links **12***ab* to **12***da*. Such use of a set of backup links is conventional.

Link **12***a* couples the output port **14***a* with the input port **16***b* of module **10***b*. Link **12***b* couples the output port **14***b* with the input port **16***c* of module **10***c*. Similarly, link **12***c* couples the output port **14***c* with the input port **16***d* of module **10***d*. Finally, link **12***d* couples the output port **14***d* with the input port **16***a* of module **10***a*. Each communication link **12** is a high-speed communication path. The capacity for links **12** is established depending on the number of modules **10** involved and the number of user devices attached to modules **10**. In the specific embodiment shown herein communication links **12** operate at 300 Mbps.

Modules **10** handle both synchronous data packets and asynchronous data packets. The synchronous packets **25** are fixed-length 16 byte packets **25**. The asynchronous packets are variable length packets **35**. Each module **10** has a number of asynchronous ports **30** (designated **30***a* to **30***d*) coupled to an asynchronous data buffer **32** and a number of synchronous ports **34** (designated **34***a* to **34***d*) coupled to a synchronous data buffer **36**. Computers **18** are connected to asynchronous data ports **30** and telephones **20** and **22** are connected to synchronous data ports **34**.

Modules **10** are interconnected by links **12**. Links **12** establish a combined synchronous and asynchronous message transmission data network whereby computers **18** may share resources and data and whereby telephone conversations may be conducted among the population of telephones **20** and **22**.

Each module **10** has a timer **37** (individually identified as timers **37***a* to **37***d*) which controls the timing within the module. Each module also includes a conventional programmed RISC processor and an associated memory which store and execute the programming operations described below.

Each module also has a plurality of user devices connected thereto. As shown in FIG. **1**, the user devices connected to the modules **10** include various computer work stations or terminals **18**, analog telephones **20**, and digital phones **22**.

Each analog telephone **20** is connected to a packet assembler and disassembler (PAD) **24**. FIG. **2** illustrates in more detail a PAD **24**. A PAD includes an analog-to-digital converter **24***a*, a sampling circuit **24***b*, a packet buffer **24***c*, and a packet assembler and disassembler block **24***d*. Each PAD **24** produces a sequence of 16 byte packets **25** representing voice sampled from a corresponding analog telephone **20**. Such a sequence of packets carry "one side" of a telephone conversation.

Each PAD **24** also receives a sequence of 16 byte packets **25** for audible presentation of voice at the corresponding analog telephone **20**. Block **24***d* drives a digital-to-analog converter **24***e* with incoming packets **25**, i.e., the "other side" of a telephone conversation involving a user and telephone **20**. Thus, block **24***d* operates within a given module **10** providing and receiving packets **25** representing an analog telephone conversation and the associated normal in-band telephone signals. Packet **25** transport occurs at 64 kbps in order to support the full duplex telephone traffic.

Digital telephones **22** produce similar packets **25** representing one side of a telephone conversation, i.e., voice sampled by digital telephone **22**, and also receive a sequence of packets **25** for audible presentation at a digital telephone **22**. Digital telephones **22** exchange packets **25** with a module **10** at sufficient speed to support a full duplex telephone conversation, i.e., 64 kbps.

Telephone conversation data from telephones **20** and **22** is packetized in the fixed-length 16 byte packets **25**. The packets containing voice data must be delivered in a timely manner in order to maintain acceptable quality of voice communication. In order to accomplish timely delivery of data representing voice communications, all such data is handled by the present invention in a synchronous fashion. Since such data is handled in synchronous fashion conventional "virtual circuits" can be established between user devices, e.g., between members of the population of tele-

phones 20 and 22. High quality telephone connections can therefore be achieved. There is no perceptible degradation in voice quality, because no more than about a four and a half millisecond delay exists in delivery of any given packet 25 from end terminal to end terminal (i.e. from telephone to telephone).

Computer work stations 18 produce data for delivery to other stations 18 and receive data from other stations 18. Because variation in delay and variation in packet size is generally acceptably in communications between stations 18, such data is managed in an asynchronous fashion when transported via modules 10. Information exchanged among stations 18 is divided into "Ethernet" type packets, i.e., variable sized packets including addressing information according to an Ethernet type addressing schemes.

The time frame for communication on links 12 is divided into a sequence of windows. FIG. 3 illustrates a sequence of windows 50, individually identified as windows 50*a*, 50*b*, etc. Each window 50 is two millisecond long

Each window 50 begins with a "start window" control character or field 52 (which is one byte long). The start window control character indicates the onset of a window 50. The remainder of each window 50 is dedicated first to all pending synchronous data transmissions and then to asynchronous data transmissions. More particularly, a first portion 54 of each window 50 is dedicated to exchange of all pending synchronous data packets 25. After all pending synchronous data packets 25 have been exchanged among modules 10, a second control character (not explicitly shown in FIG. 3) is transmitted around the ring to indicate the end of the synchronous transmissions. A second portion 56 of each window 50 is dedicated to exchange of asynchronous data packets 35. At the end of each window another control character (not explicitly shown in FIG. 3) is transmitted around the ring. As will be explained later, tokens and timing characters are also transmitted around the ring.

The length of window 50 is two milliseconds long. The length of windows 50 is established by taking into account the bandwidth of the various communication paths. The length of window 50 is established so that all synchronous data can be delivered during each window and so that after the synchronous data is transmitted, sufficient reserve will remain in each window 50 to conduct exchange of asynchronous data. The actual allocation of a given window 50 between synchronous and asynchronous data is dynamic. The allocation depends on the amount of pending synchronous data packets 25 which must be transmitted during the first portion of a given window 50. As the number of telephone conversations increases, the portion 54 of window 50 used for such conversations increases.

Thus the allocation between synchronous and asynchronous data in a given window 50 is not fixed but rather a function of the amount of synchronous data pending at the beginning of the window 50 with the remaining portion 56 being used for asynchronous data. A control character 53 which indicates that synchronous traffic is "all done" separates portions 54 and 56 of each window. This control character indicates the end of synchronous data transmission and the beginning of asynchronous data transmission within a given window 50.

FIG. 4 illustrates, at a system level, data exchange during a given window 50. As shown in FIG. 4, processing loops at decision block 100 until start window control character 52 appears on links 12. Processing then advances to block 102 where modules 10 exchange all pending synchronous data packets, i.e., deliver all pending packets 25 in the synchronous data buffers 36. Block 102 represents the overall exchange of all pending synchronous data packets 25 among modules 10*a*–10*d* by ring message exchange.

As indicated by decision block 106 a determination is made that all modules 10 have completed exchange of pending synchronous data, e.g., all modules 10 have delivered all pending synchronous data packets 25. In other words, all synchronous data in buffers 36 at the onset of the current window 50 have been transmitted via links 12 to an appropriate module 10. At this point, communication among modules 10 switches from a synchronous mode of operation to an asynchronous mode of operation allowing variable length packets and an alternate addressing scheme. Asynchronous transmission is done using Ethernet packet rules and addressing codes to route the variable length packets to particular modules 10 and to corresponding user devices attached thereto. Ethernet packaging rules allow packets of varying length between 64 and 1524 bytes

To prevent monopolization of window 50 by one module, each module 10 limits its use of portion 56 of each window so as to allow other modules 10 to deliver asynchronous data. Thus, block 108 represents delivery of a limited amount of asynchronous packets followed in decision block 110 which tests use limitations. During block 108, a module 10 sends a certain number of Ethernet packets to a successive one of modules 10. Such module 10 limits its further use of the asynchronous portion 56 of a given window 50. That is, each module 10 is allowed a limited number of asynchronous data bytes per given window 50. In the embodiment described herein, each module 10 transmits a maximum of 4000 bytes of asynchronous data in any given window 50. Thus, system level operation loops at blocks 108 and 110 until all modules 10 have reached their use limit for the current window 50 or have delivered all pending asynchronous data. Processing eventually returns to block 100 where system 8 waits for occurrence of the start window control character 52 and a next window 50.

The modules 10 in general operate on a "peer" basis. However, one of modules 10 is given some degree of control over the process. In the embodiment shown, module 10*a* executes master control, that is, to some extent module 10*a* orchestrates the exchange of information on links 12 and it is in effect a timing master for the system. Modules 10 other than module 10*a* may be inserted and removed from the system as needed or desired without corrupting an overall control strategy, However, there must always be a control module 10*a*.

Master control module 10*a* makes use of control characters and tokens to orchestrate packetized information exchange within system 8. Modules detect the receipt of a token or control character by detecting one of the decodes of a byte other than the 256 data decodes. When a module 10 receives a control character or a timing character, it immediately retransmits the control character or timing character to the next module 10. When a module 10 receives a token, the token is held until the module 10 is ready to retransmit the token, i.e. until the module is ready to relinquish its right to send packets of a particular type.

FIG. 5 illustrates programming with respect to operation of module 10*a*. Module 10*a* is the master control module. As indicated by block 510, the process begins when module 10*a* transmits a "window start" control character 52 (see FIG. 3). Control character 52 is sent immediately around the ring architecture because when a module 10 receives a control character it immediately re-transmit (i.e. repeats) the control character. At this point, all modules 10 are prepared for the

onset of a window **50**. As indicated by block **511** module **10***a* transmits all its pending synchronous packets **25**, i.e., module **10***a* empties synchronous data buffer **36***a*, into the ring on communication link **12***a*. Once module **10***a* has transmitted all of its synchronous data packets **25**, as indicated by block **512** module **10***a* passes the "transmit synchronous packets" token to the next module, i.e., to module **10***b*.

After passing the transmit synchronous packets token to the next module on the ring, module **10***a* processes synchronous packet traffic from other modules until the transmit synchronous packets token is returned to module **10***a*. This is indicated by blocks **513** and **514**. During this time, the remaining modules **10** will each in turn have opportunity to send all synchronous data packets **25** which were pending at the onset of the current window **50**. That is, after module **10***b* receives the token, it transmits all its pending synchronous data packets **25**, i.e., empties synchronous data buffer **36***b*, onto link **12***b*. Module **10***b* then passes the token to module **10***c*, giving module **10***c* an opportunity to send all its synchronous data packets **25** onto link **12***c*. The token is then passed to module **10***d*. When module **10***d* receives the token, it in turn submits all its synchronous data packets **25** which were pending at the onset of the current window **50** onto link **12***d*. As each module **10** submits its synchronous data packets **25** onto the ring architecture, each packet **25** reaches a target or module **10** which has attached thereto one of telephones **20** or **22** so as to complete a virtual circuit.

Eventually, all modules **10** will have had an opportunity to submit synchronous data packets **25** onto the ring and the "transmit synchronous packets" token will return to module **10***a*. Processing then advances from decision block **513** to block **516** and module **10***a* sends the "all done synchronous data" control character **53** out on link **12***a*. Each of modules **10***b*–**10***d* thereby receive the "all done synchronous data" control character **53** indicating a transition from synchronous data exchange to asynchronous data exchange. As indicated by blocks **517** and **518**, module **10***a* transmits asynchronous data packets **35** onto link **12***a*. As may be appreciated, each of the asynchronous data packets pass around the ring and eventually reach the intended destination, i.e., one of modules **10***b*–**10***d* addressed as the destination address in the packet.

Module **10***a* stops sending asynchronous data packets when one of two conditions is satisfied as indicated by decision block **517**. Transmission of asynchronous packets by module **10***a* stops when module **10***a* determines that it has no more asynchronous data to transmit, (i.e., asynchronous data buffer **32***a* is empty) or if module **10***a* has reached its use limit. In defined a module is limited to transmitting 4000 bytes in one session. If module **10***a* has not reached its use limit and if there are additional asynchronous data packets **35** in buffer **32***a*, then processing returns to block **518** and module **10***a* continues to transmit asynchronous data packets Eventually, module **10***a* either reaches its use limit or exhausts pending asynchronous data in buffer **32***a*. Processing then advances to block **520** and module **10***a* passes the token to the next module, i.e., to module **10***b*.

After passing the token to the next module, module **10***a* will process asynchronous data traffic as indicated by blocks **521** and **522**. Module **10***a* checks for return of the token as indicated by decision block **421**. That is processing as indicated by blocks **521** and **522** continues until the token is returned to module **10***a*. When the token returns to module **10***a*, all modules **10** have had opportunity to send asynchronous data packets **35** onto the ring at least once up to their given limit, i.e., at least 4000 bytes.

At this point, some portion of window **50** may remain. This is determined as indicated by block **523**. If more time remains module **10***a* can take advantage of this opportunity to send more asynchronous data packets **35**. As indicated in FIG. **5** The "no" output from block **523** goes back to block **517** and the process repeats.

Eventually, module **10***a* runs out of time in the current window **50** for transmission of additional packets **35**. An end of ring control character is then transmitted around the ring. Processing then returns to block **510** where module **10***a* again sends control character **52** and the process repeats.

Module **10***a* also provides a timing reference for the system. Module **10***a* includes an interval timer **37***a* which produces interrupt signal every 15.625 microseconds. This timing reference signal is transmitted from module **10***a* to the other modules in the ring. When the timing interrupt occurs, module **10***a* transmits the special timing control character. The timing control character is inserted into any packet that is being transmitted at the time the interrupt occurs. Thus, some synchronous packets **25** may be 17 bytes long after the clock control character is inserted therein. The additional delay introduced, i.e., a 16 byte synchronous packet **25** versus a 17 byte synchronous packet **25**, does not introduce any noticeable delay to persons engaged in a conversation. The timing character is a 10 bit character which is not used for any other purpose and which each units recognizes as a timing character. When a unit on the ring (other than module **10***a*) detects this character, it repeats the character to the next unit on the ring and at the same time it re-synchronizes its internal clock **37**. That is, the clock **37** in each unit is re-synchronized when the timing character is detected. In this way the clocks **37** in the various modules are kept in close synchronization. It is noted that as shown herein it is the control module **10***a* which introduces the timing character onto the ring, any one of the modules could perform this function.

FIG. **6** illustrates the programming for modules **10** other than module **10***a*. (FIG. **5** shows the programming for module **10***a*). That is, FIG. **6** illustrates programming for modules **10***b*–**10***d*. The process starts as indicated by block **610** when a module receives a "start window" control character **52**. After receiving a "start window" control character a module looks for a token with indicates that the module should start transmitting synchronous data. Processing continues iteration between blocks **611** and **612** until the module **10** receives the "start transmitting synchronous data" token. As indicated by block **611**, when a module receives the "start transmitting synchronous data" token the module begins transmission of its synchronous data. Once a module **10** has transmitted all pending synchronous data packets **25** into the ring, it passes the "start transmitting synchronous data" token to the next module **10**. This gives the next module **10** opportunity to submit its synchronous data packets **25**.

Once a module **10** has passed the "start transmitting synchronous data" the module processes synchronous packet traffic from other modules as indicated by block **615**. Decision block **614** indicates that a module looks for "all synchronous traffic done" control character **53**. Until the "all synchronous traffic done" control character **53** appears, processing iterates at blocks **614** and **615** and the module processes any synchronous data packets **25** appearing in the ring architecture from other modules. When the "all synchronous traffic done" control character **53** does appear processing advances to block **616** and **617** where the module processes any asynchronous packet traffic appearing in the ring architecture from other modules. In decision block **616**, module **10** looks for the "send asynchronous data" token. Processing iterates at blocks **616** and **617** until the module

receives the "send asynchronous data" token. "send asynchronous data" token is received asynchronous data packet 35 are transmitted as indicted by block 620. After transmitting each asynchronous data packet 35, the module determines as indicated by block 619 whether it has any additional asynchronous data packets to transmit, or if the module has reached its use limit, e.g., has transmitted 4000 bytes of asynchronous information in the current window 50. Processing iterates at blocks 619 and 620 until no further asynchronous data packets remain or until the module 10 has reached its allotment or use limit allowed in the current window 50. Processing then advances to block 621 and the module passes the "send asynchronous data" token to the next module 10.

After each of the modules 10 has had an opportunity to transmit both synchronous and asynchronous traffic during a particular window, there may still be time remaining in the window. When this condition occurs, the modules are given another chance to transmit additional asynchronous packets. This condition is illustrated in FIG. 6 by the path from block 622 through block 625 to the entry of block 619.

Following block 621, processing iterates between blocks 622 and 623 where the module processes any further asynchronous packet traffic from other modules and looks for the occurrence of the end of window and the start new window control characters. If an end of window and start of new window control characters have not been received the processing goes from block 622 to 625 to 619. When and end of window and start of new window control characters are received the processing returns to block 611.

As an example of the capacity of the system, it is noted that a voice switching capacity on the order of 256 simultaneous, full duplex calls may be implemented on a stack of eight modules 10. Each full duplex voice call consumes 64 kbps of data bandwidth. This translates into: (64,000*2)*256=32,768,000 or 32 Megabits (Mbps) of voice switching bandwidth on links 12 to support 256 simultaneous full duplex voice calls. This is exclusive of framing overhead, which will be dependent on the hardware implementation.

There are also other capacity considerations. Services which use 'redirection', e.g., voice compression, voice recognition or fax services, all are very processor bandwidth intensive. This component creates an 'overhead factor' on the base voice switching bandwidth independent of framing overhead. Using an estimated overhead factor of thirty-three percent, the voice switching bandwidth requirement increases from about 32 Mbps to about 44 Mbps.

For asynchronous data capacity, a minimum carrying requirement of a single 100 Mbps Ethernet will meet a given level of computer network use expectations. This brings the aggregate carrying capacity for system 8 to about 144 Mbps.

For control and management an overhead of not more than about 4 Mbps per module 10 is expected. Of this, 1 Mbps is reserved for true inter-module 10 communication, 0.5 Mbps is reserved for network management, 1.5 Mbps is reserved for call accounting and 1 Mbps is reserved for event logging and tracing. For eight modules 10 in a system 8, a subtotal of (8*4,000,000) or about 32 Mbps. This brings the entire switching capacity requirement for system 8 to about 176 Mbps. Establishing a 300 Mbps capacity for links 12 supports this expected switching capacity.

Naturally, other systems could be implemented using the present invention with different requirements and capacity considerations. For example various numbers of modules 10 could be connected in a ring utilizing the present invention, While the invention is described as applied to a LAN environment, it is noted that the invention could also be applied in a WAN environment. It is also noted that the communication path between modules 10 could be either electrical or optical without departing from the present invention.

It should be appreciated that the computer data communicated between modules 10, could include all the various kinks of data normally transmitted between computer. For example such data could include video data and PC-telephone data.

A combined synchronous and asynchronous message transmission method and apparatus has been shown and described. The integration of synchronous and asynchronous message transmission into a single communication system provides opportunity for integrated communication services incorporating both computer data and voice data. Despite integration of synchronous and asynchronous data, synchronous data arrives in timely fashion without degradation in voice quality.

As described herein data from computer 18 is treated as asynchronous data. However, under certain conditions, computers 18 may be required to deliver time-sensitive information in a synchronous manner, and could be treated as such by a corresponding module 10.

System 8 also includes an external connection 38, e.g., a high speed telephone or network connection, whereby other systems may introduce information into system 8 or take information from system 8. Such links are handled similar to links directly connected to modules 10.

It is noted herein that a single asynchronous protocol is used. However, additional control schemes may also be employed. For example such protocols could be used to allow further exchange of asynchronous data when modules 10 have reached their use limit for asynchronous data, but window 50 has not yet expired.

It will be appreciated that the present invention is not restricted to the particular embodiment that has been described and illustrated, and that variations may be made therein without departing from the scope of the invention as found in the appended claims and equivalents thereof.

What is claimed is:

1. A communication system comprising:

a plurality of communication modules interconnected to exchange information there between,

said information including a variable amount of synchronous information and a variable amount of asynchronous information,

said modules providing and receiving said synchronous information in substantially fixed length packets, said modules providing and receiving said asynchronous information in variable length packets, each of said packets comprising a plurality of multi bit bytes;

a communication channel coupling said modules together, operation of said communication channel being divided into repeated fixed length windows, said channel operating in accordance with a token passing protocol,

each module transmitting a variable number of fixed length packets between said modules during a first portion of each of said fixed length windows and each module transmitting a variable number of variable length packets between said modules during a second portion of each of said repeated fixed length windows,

each module transmitting a token to the next module when that module has completed the transmission of all

synchronous packets from that module to other modules, in order to indicate that the next module can begin transmission of synchronous packets, and

a timer which periodically inserts a timing byte into the string of bytes being transmitted between said modules on said communication channel, each of said modules including a timer which is resynchronized in accordance with said timing byte.

2. A system for transmitting synchronous and asynchronous information, said system comprising:

a plurality of modules coupled in a ring, each module receiving input from a predecessor module and providing output to a successor module, each of said modules referencing a repeating sequence of time windows, each time window including a first portion dedicated to exchange of synchronous data and a second portion dedicated to exchange of asynchronous data, said ring operating in accordance with to a token passing protocol, said synchronous data being in the form of fixed length packets and said asynchronous data being in the form of variable length packets,

said first portion of said time window terminating when all pending synchronous data has been exchanged among said plurality of modules, each module transmitting a token to the next ring when it is completed transmitting all synchronous data pending in that module, use of said second portion of said time window by each module being limited whereby each of said modules has opportunity during said second portion of said window to transmit some asynchronous data,

each module including a clock, synchronization between said modules being maintained by periodically transmitting a character on said ring which is recognized by each module as a timing character and wherein each modules resynchronizes its clock when said character is received.

* * * * *

US005602841A

United States Patent [19]

Lebizay et al.

[11] Patent Number: 5,602,841

[45] Date of Patent: Feb. 11, 1997

[54] **EFFICIENT POINT-TO-POINT AND MULTI-POINT ROUTING MECHANISM FOR PROGRAMMABLE PACKET SWITCHING NODES IN HIGH SPEED DATA TRANSMISSION NETWORKS**

[75] Inventors: **Gerald Lebizay**, Vence; **Jean M. Munier**, Cagnes sur Mer; **Andre Pauporte**, La Colle sur Loup; **Victor Spagnol**, Cagnes sur Mer, all of France

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **404,800**

[22] Filed: **Mar. 15, 1995**

[30] **Foreign Application Priority Data**

Apr. 14, 1994 [EP] European Pat. Off. 94480030

[51] **Int. Cl.**[6] **H04L 12/56**

[52] **U.S. Cl.** **370/413**

[58] **Field of Search** 370/58.2, 60, 60.1, 370/61, 79, 94.1, 94.2

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,365,519	11/1994	Kozaki et al.	370/60
5,393,280	2/1995	Zheng	370/60

FOREIGN PATENT DOCUMENTS

0575281	12/1993	European Pat. Off.	H04L 12/18
0579567	1/1994	European Pat. Off.	H04L 12/56
0632625	1/1995	European Pat. Off.	H04L 29/06
9102420	2/1991	WIPO	H04L 12/56

OTHER PUBLICATIONS

Globecom '91, vol. 1, 12–92 NY pp. 104–110.
Int. Journal of Digital and Analog Communication Systems, vol. 5, 1992 pp. 29–37.
Proceedings of the IEEE '92 Custom Integrated Circuits Conf., 5–29, NY, pp. 14.4.1–14.4.4.

Primary Examiner—Douglas W. Olms
Assistant Examiner—Russell W. Blum
Attorney, Agent, or Firm—Stephen T. Keohane; John J. Timar

[57] **ABSTRACT**

The present invention relates to an efficient point-to-point and multi-points routing system and method for programmable data communication adapters in packet switching nodes of high speed networks. The general principles of this efficiency are the following:

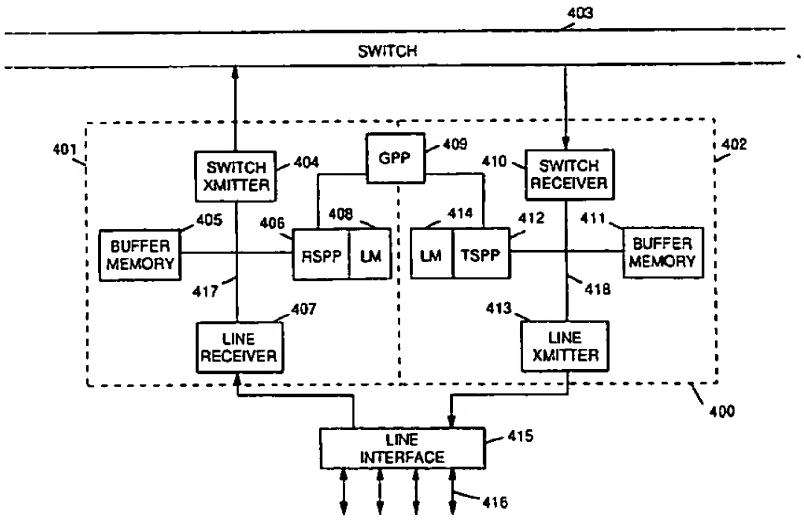
First, data packets are never copied, only packet pointers are copied for each destination: Space in Buffer Memory is saved, the number of instructions is significantly reduced improving the packet throughput (number of packets per seconds that the adapter is able to transmit). and the routing is independant of the packets length.

Second, no overhead is generated by the multi-points mechanism in the real time procedures: the underrun/overrun problems on the ouputs are reduced and the efficiency of the adapter in term data throughput (bits per second) is significantly improved.

Third, each output is processed independently by means of interrupts: lines are managed in real time and lines of different speed or protocol can be supported in parallel.

Fourth, the release of the resources is entirely realized on a non priority mode.

19 Claims, 19 Drawing Sheets



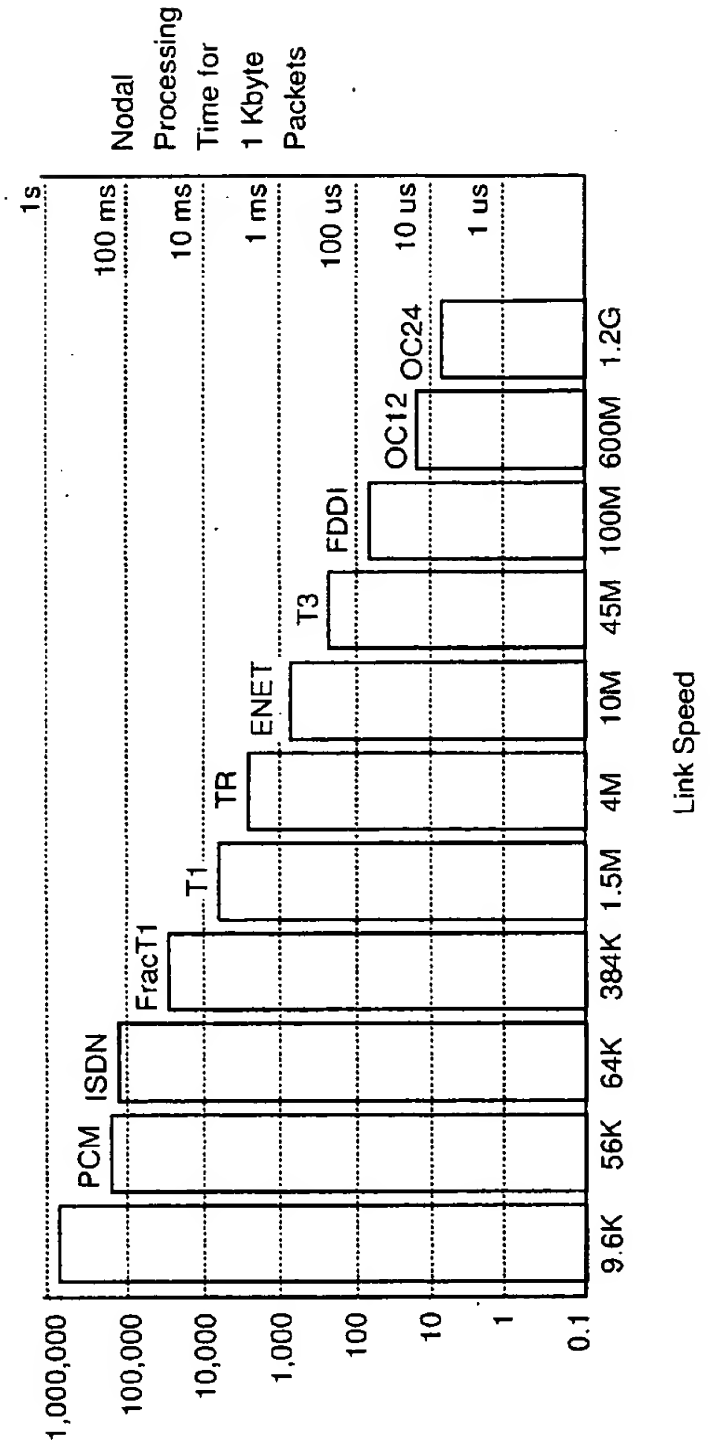

Nodal Processing Time for 1 Kbyte Packets
1s
100 ms
10 ms
1 ms
100 us
10 us
1 us
PCM
ISDN
FracT1
T1
TR
ENET
T3
FDDI
OC12
OC24
9.6K
56K
64K
384K
1.5M
4M
10M
45M
100M
600M
1.2G
Link Speed
1,000,000
100,000
10,000
1,000
100
10
1
0.1
Number of Instructions at 1 Mips

FIG 1

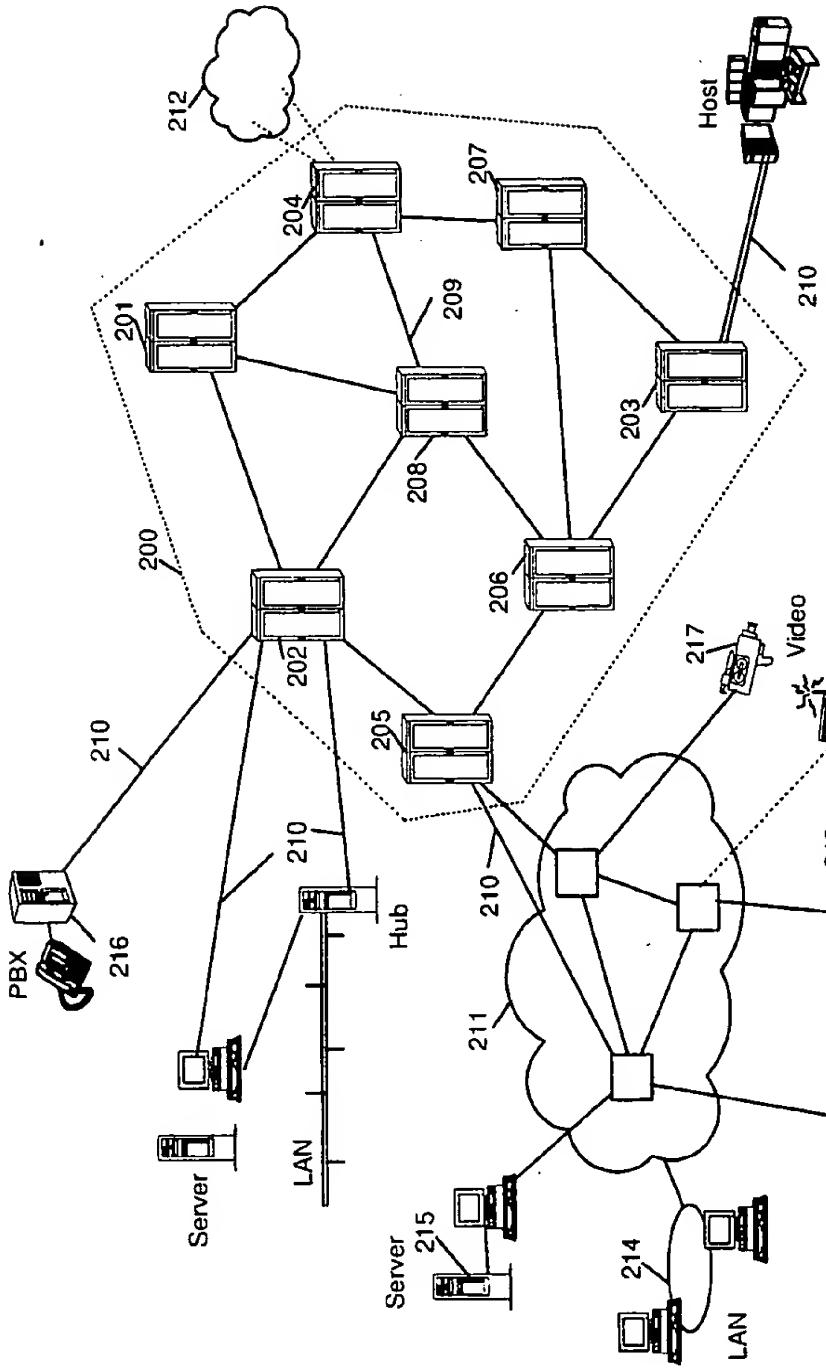

212
Host
207
204
210
209
201
203
208
200
206
202
217
Video
Wireless
205
210
213
210
Hub
PBX
216
Host
211
Server
LAN
PBX
216
Server
215
214
LAN

FIG 2

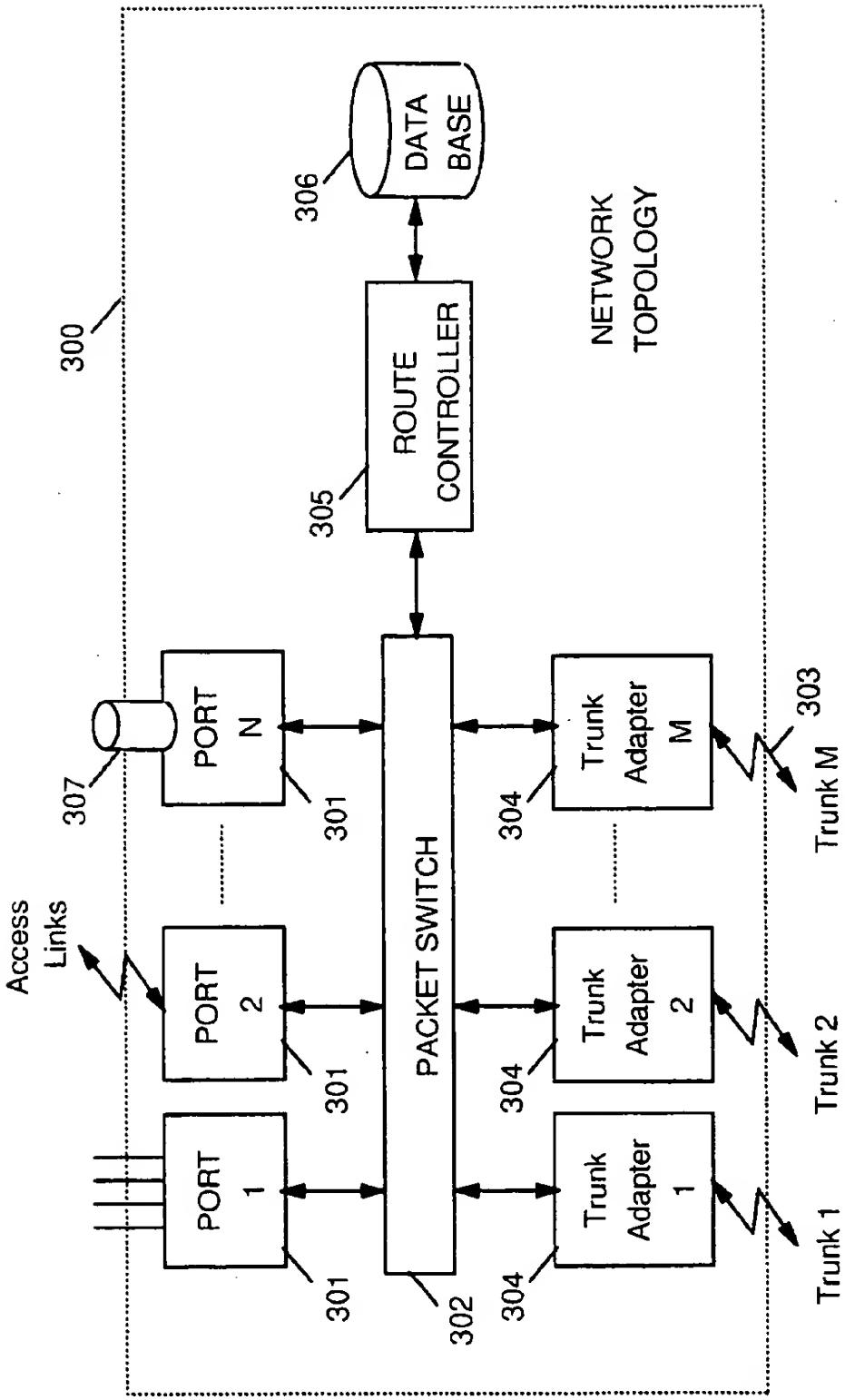

NETWORK TOPOLOGY
300
DATA BASE
306
ROUTE CONTROLLER
305
PACKET SWITCH
302
PORT N
PORT 2
PORT 1
301
307
Access Links
Trunk Adapter M
Trunk Adapter 2
Trunk Adapter 1
304
303
Trunk M
Trunk 2
Trunk 1

FIG 3

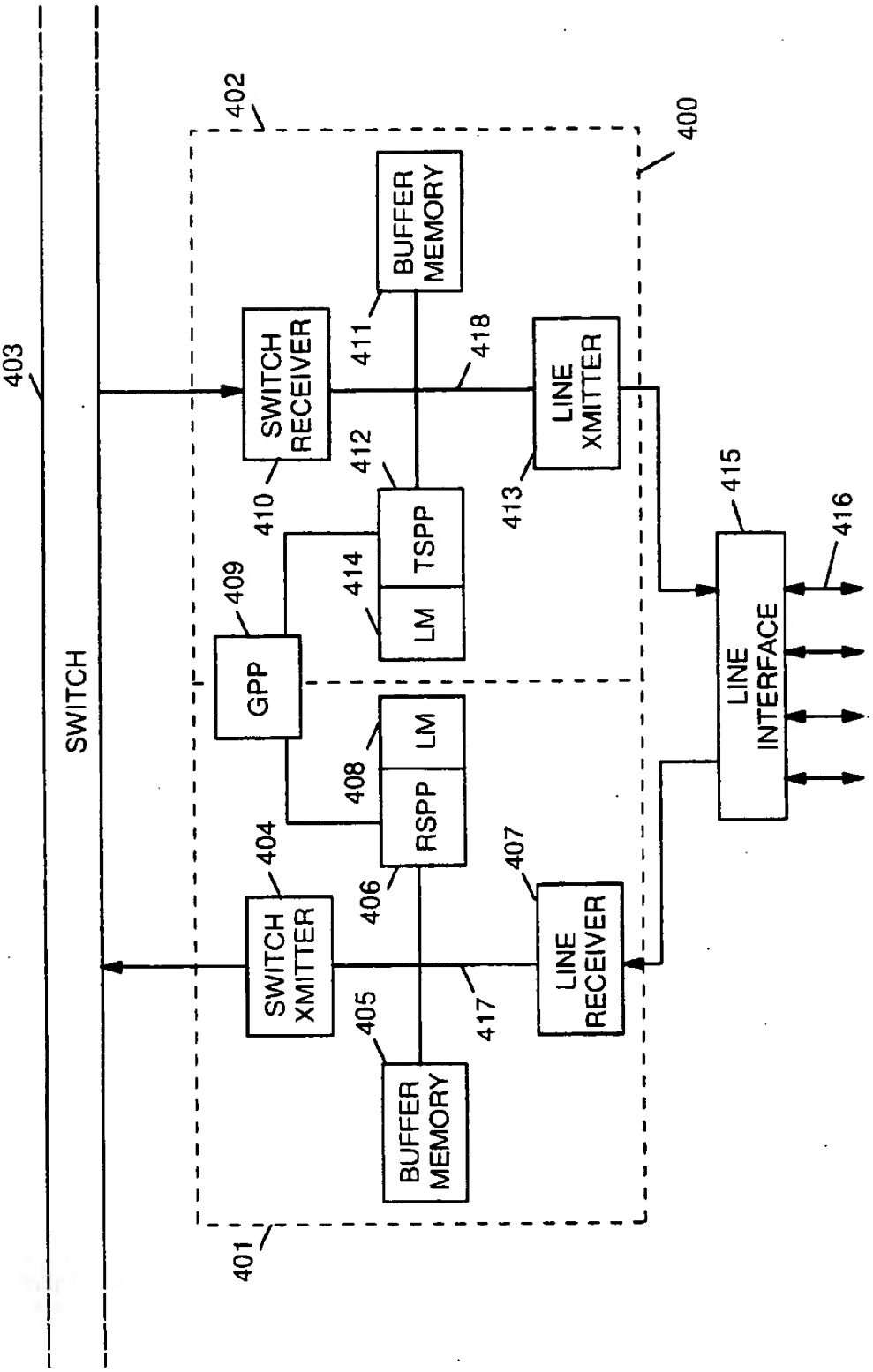

SWITCH
403
402
400
401
BUFFER MEMORY
411
SWITCH RECEIVER
410
418
LINE XMITTER
413
412
TSPP
LM
414
409
GPP
408
LM
RSPP
406
404
SWITCH XMITTER
405
BUFFER MEMORY
407
LINE RECEIVER
417
415
LINE INTERFACE
416

FIG 4

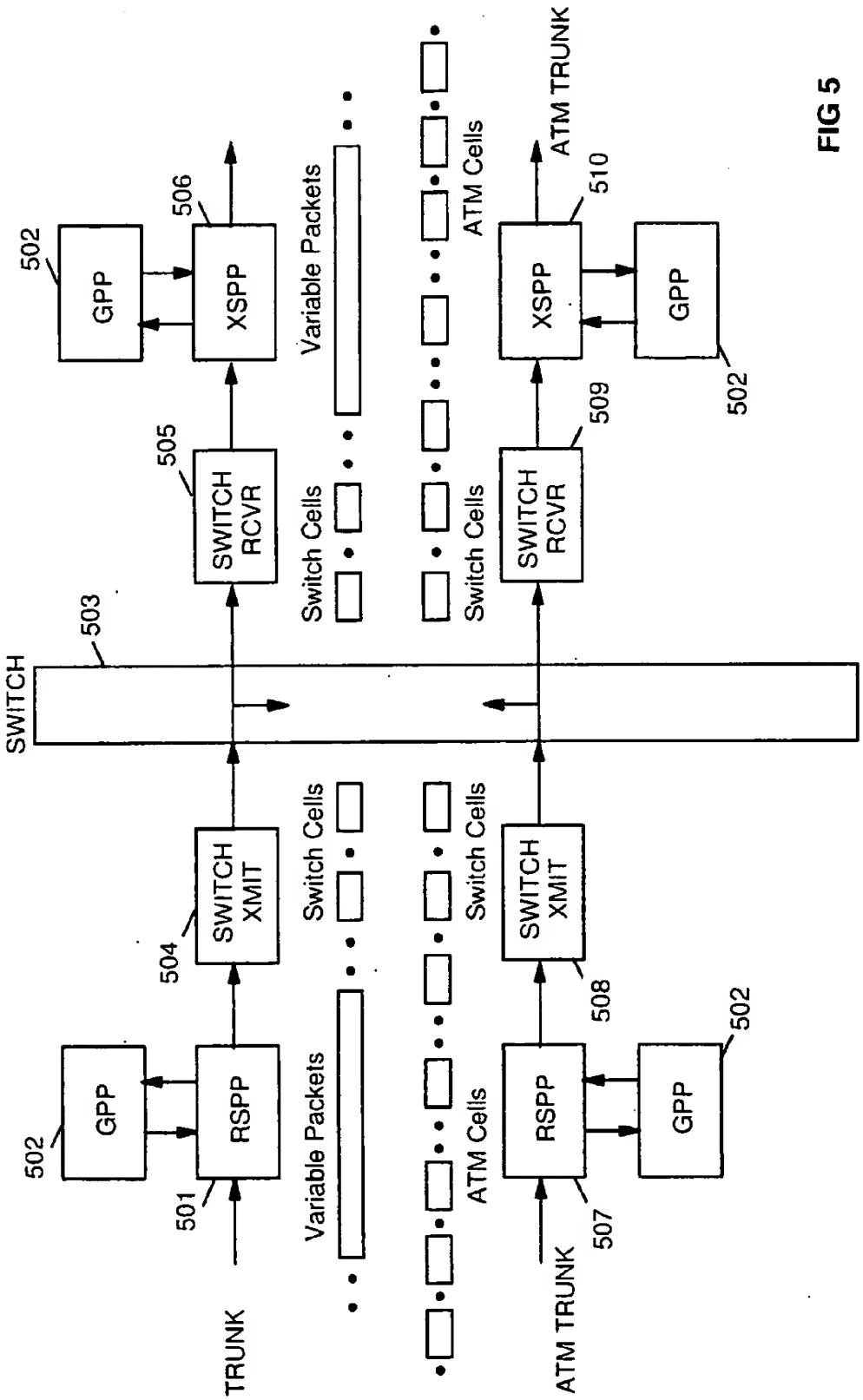

TRUNK
ATM TRUNK
501
507
502
GPP
RSPP
504
508
SWITCH XMIT
503
SWITCH
505
509
SWITCH RCVR
506
510
XSPP
ATM TRUNK
Variable Packets
Switch Cells
ATM Cells

FIG 5

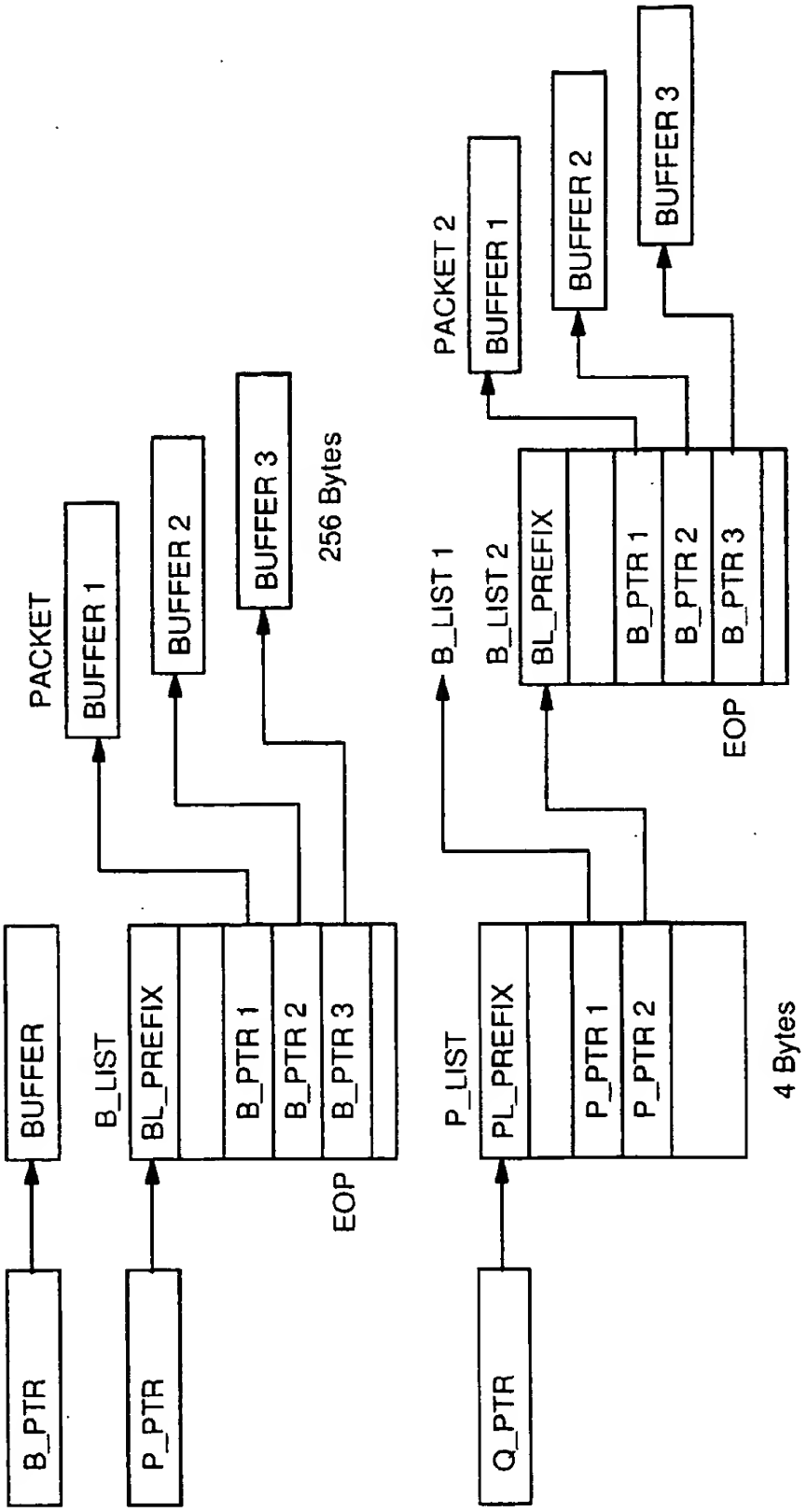

B_PTR
BUFFER
P_PTR
B_LIST
BL_PREFIX
B_PTR 1
B_PTR 2
B_PTR 3
EOP
PACKET
BUFFER 1
BUFFER 2
BUFFER 3
256 Bytes
Q_PTR
P_LIST
PL_PREFIX
P_PTR 1
P_PTR 2
4 Bytes
B_LIST 1
B_LIST 2
BL_PREFIX
B_PTR 1
B_PTR 2
B_PTR 3
EOP
PACKET 2
BUFFER 1
BUFFER 2
BUFFER 3

FIG 6

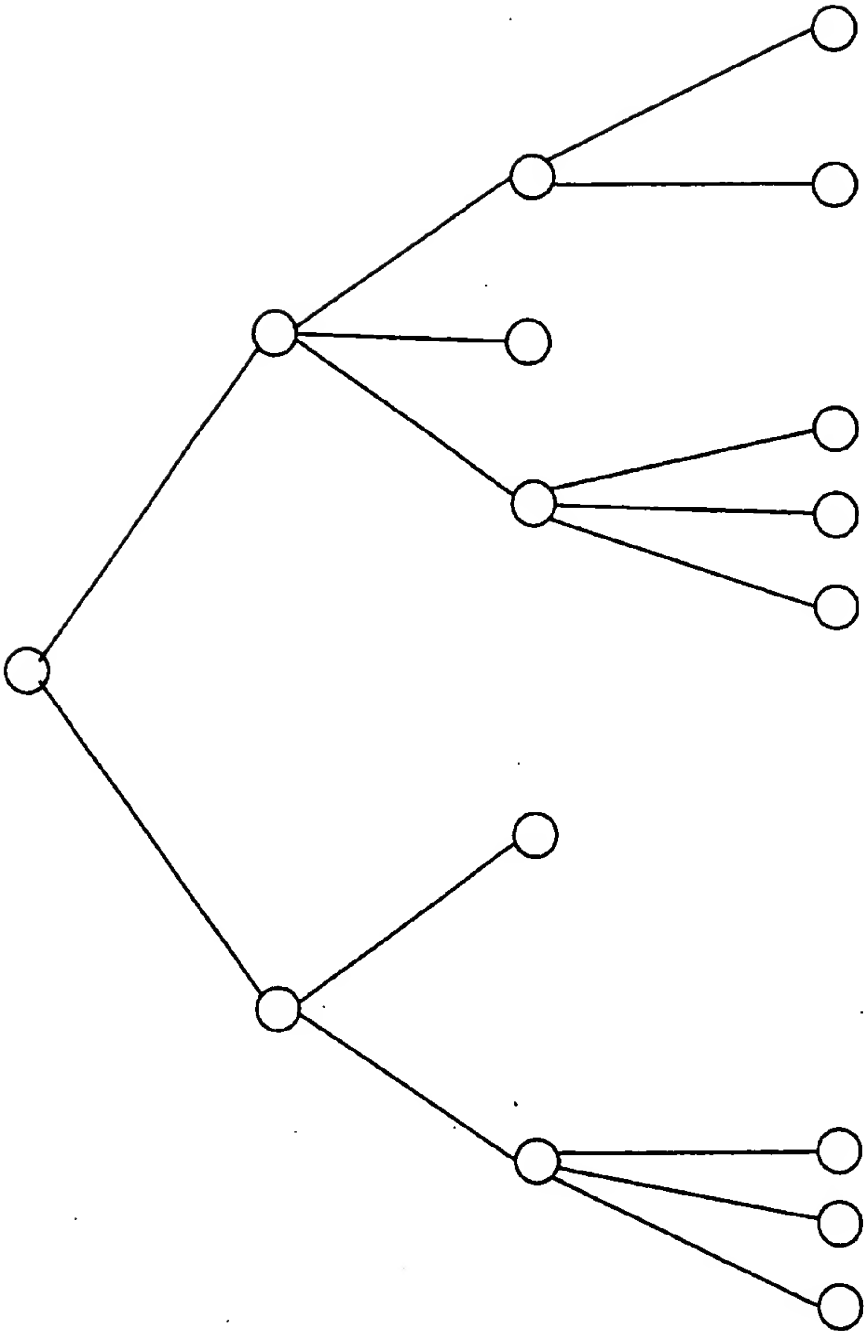


FIG 7

FIG 8A

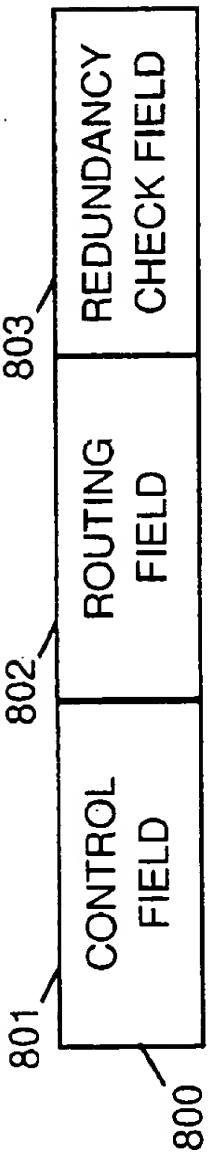
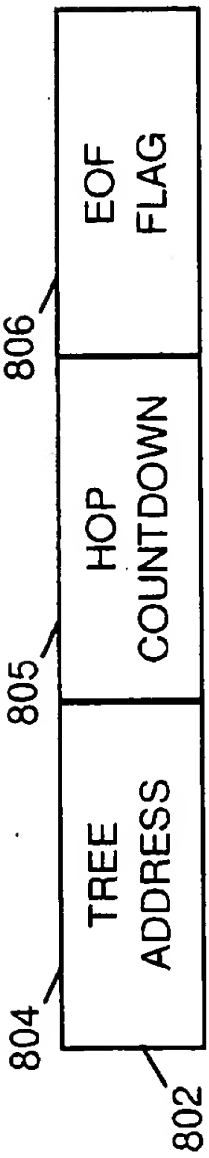

CONTROL FIELD
ROUTING FIELD
REDUNDANCY CHECK FIELD
801
802
803
800

FIG 8B


TREE ADDRESS
HOP COUNTDOWN
EOF FLAG
804
805
806
802

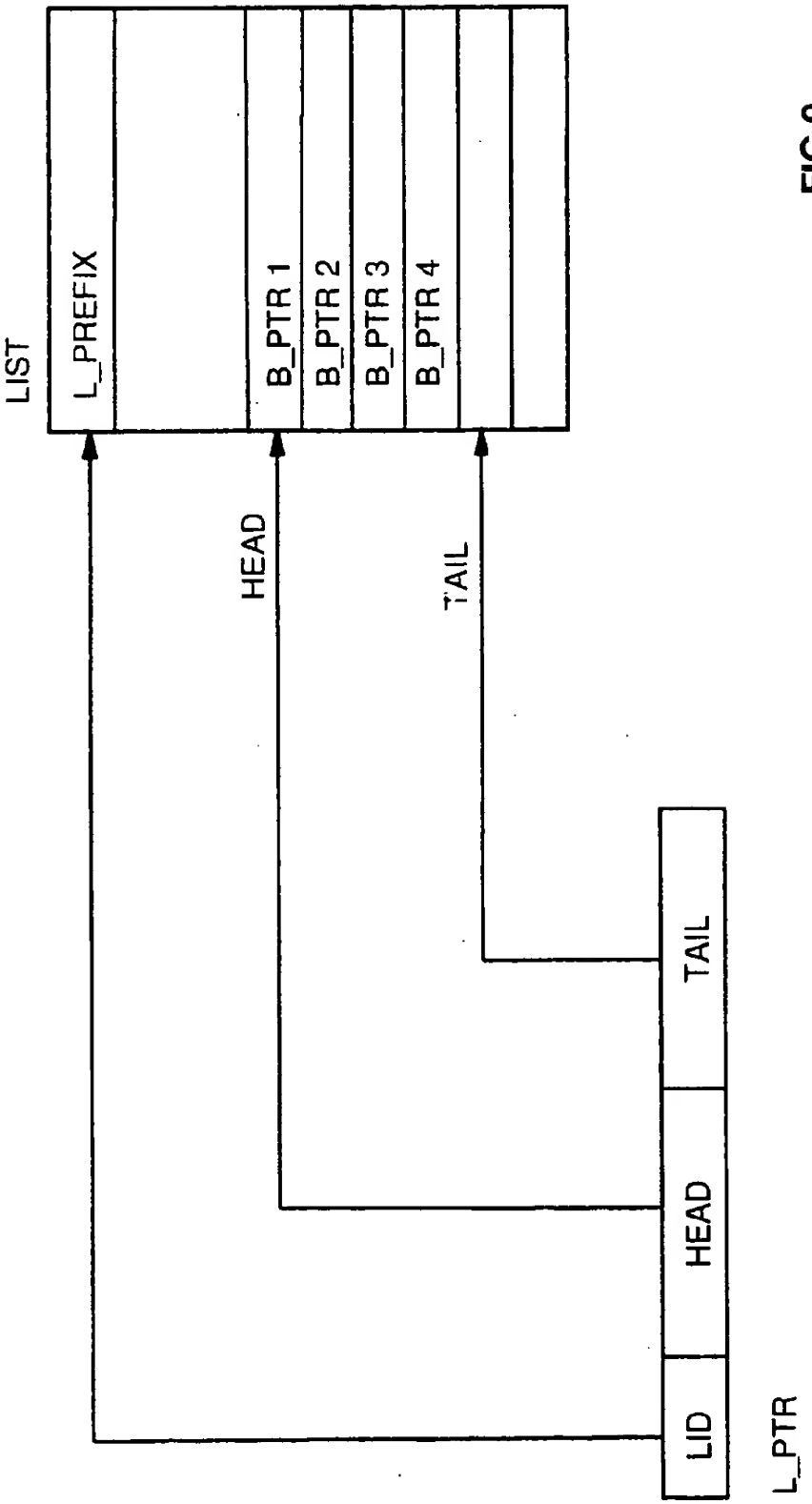

LIST
L_PREFIX
B_PTR 1
B_PTR 2
B_PTR 3
B_PTR 4
HEAD
TAIL
TAIL
HEAD
LID
L_PTR

FIG 9

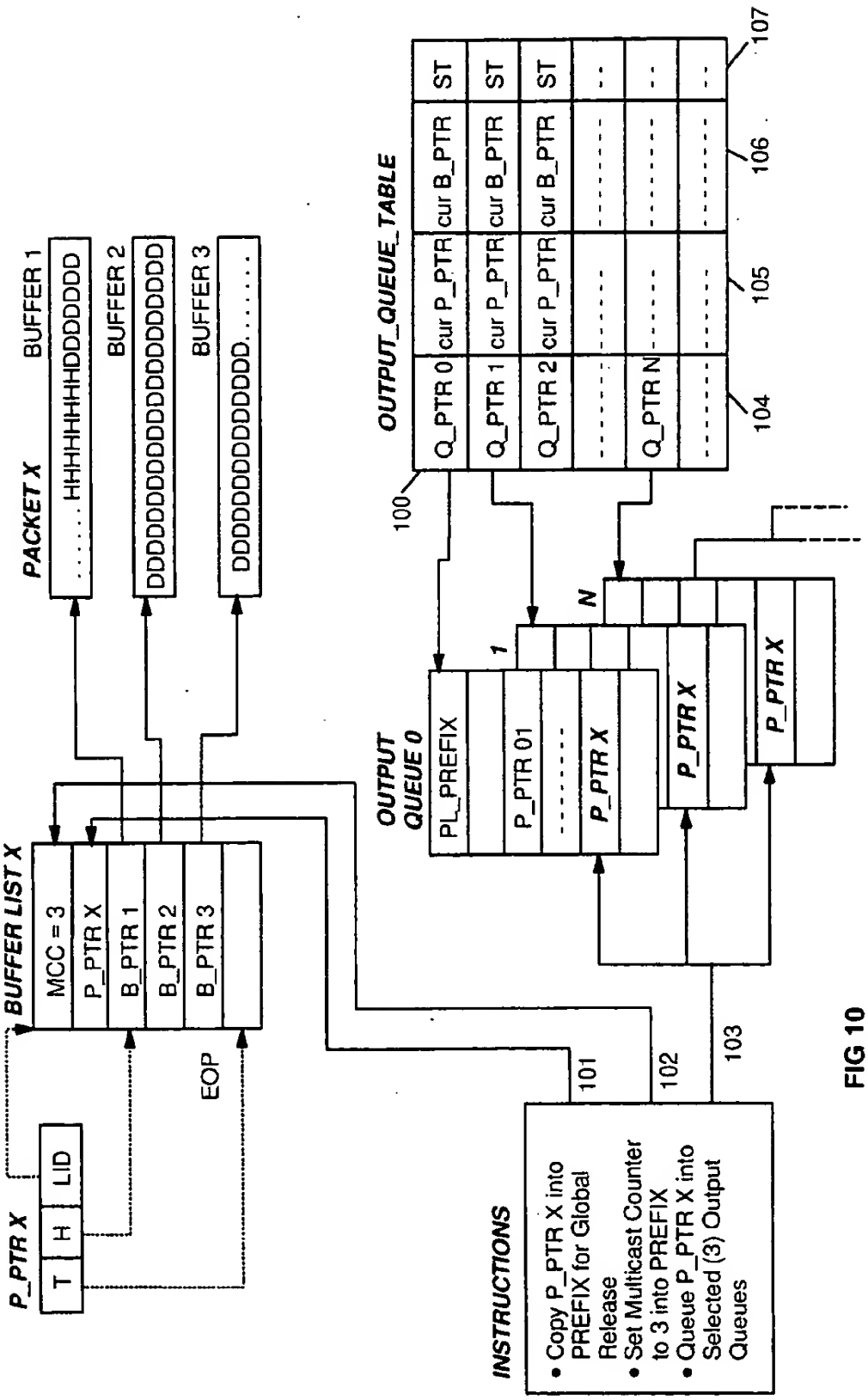

PACKET X
BUFFER 1
HHHHHHHHDDDDDDDD
BUFFER 2
DDDDDDDDDDDDDDDD
BUFFER 3
DDDDDDDDDDDD
OUTPUT_QUEUE_TABLE
Q_PTR 0
Q_PTR 1
Q_PTR 2
Q_PTR N
cur P_PTR
cur B_PTR
ST
100
104
105
106
107
OUTPUT QUEUE 0
PL_PREFIX
P_PTR 01
P_PTR X
1
N
BUFFER LIST X
MCC = 3
P_PTR X
B_PTR 1
B_PTR 2
B_PTR 3
EOP
P_PTR X
T
H
LID
INSTRUCTIONS
• Copy P_PTR X into PREFIX for Global Release
• Set Multicast Counter to 3 into PREFIX
• Queue P_PTR X into Selected (3) Output Queues
101
102
103

FIG 10

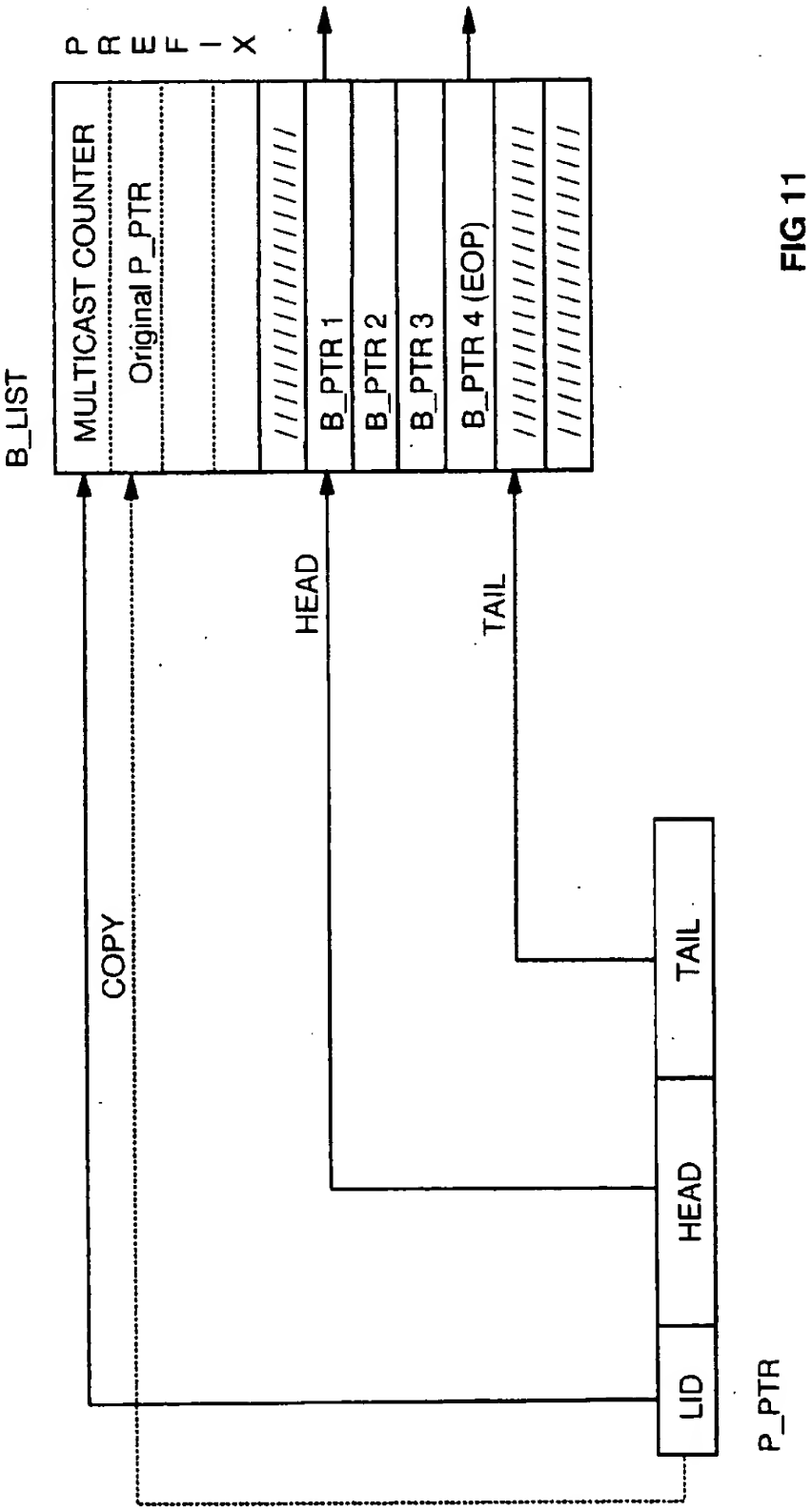

B_LIST
MULTICAST COUNTER
Original P_PTR
PREF-X
B_PTR 1
B_PTR 2
B_PTR 3
B_PTR 4 (EOP)
HEAD
TAIL
COPY
TAIL
HEAD
LID
P_PTR

FIG 11

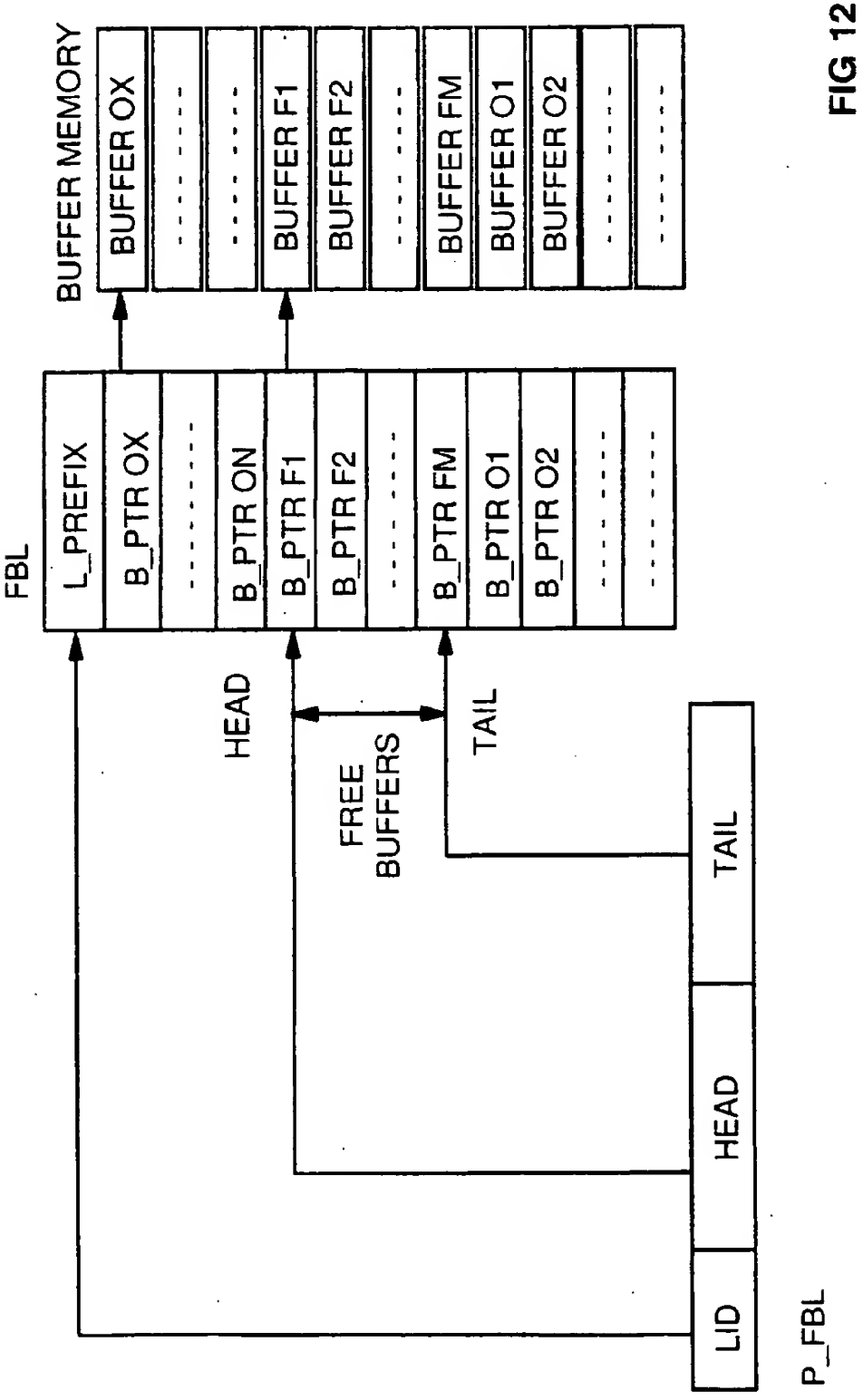

BUFFER MEMORY
BUFFER OX
BUFFER F1
BUFFER F2
BUFFER FM
BUFFER O1
BUFFER O2
FBL
L_PREFIX
B_PTR OX
B_PTR ON
B_PTR F1
B_PTR F2
B_PTR FM
B_PTR O1
B_PTR O2
HEAD
TAIL
FREE BUFFERS
TAIL
HEAD
LID
P_FBL

FIG 12

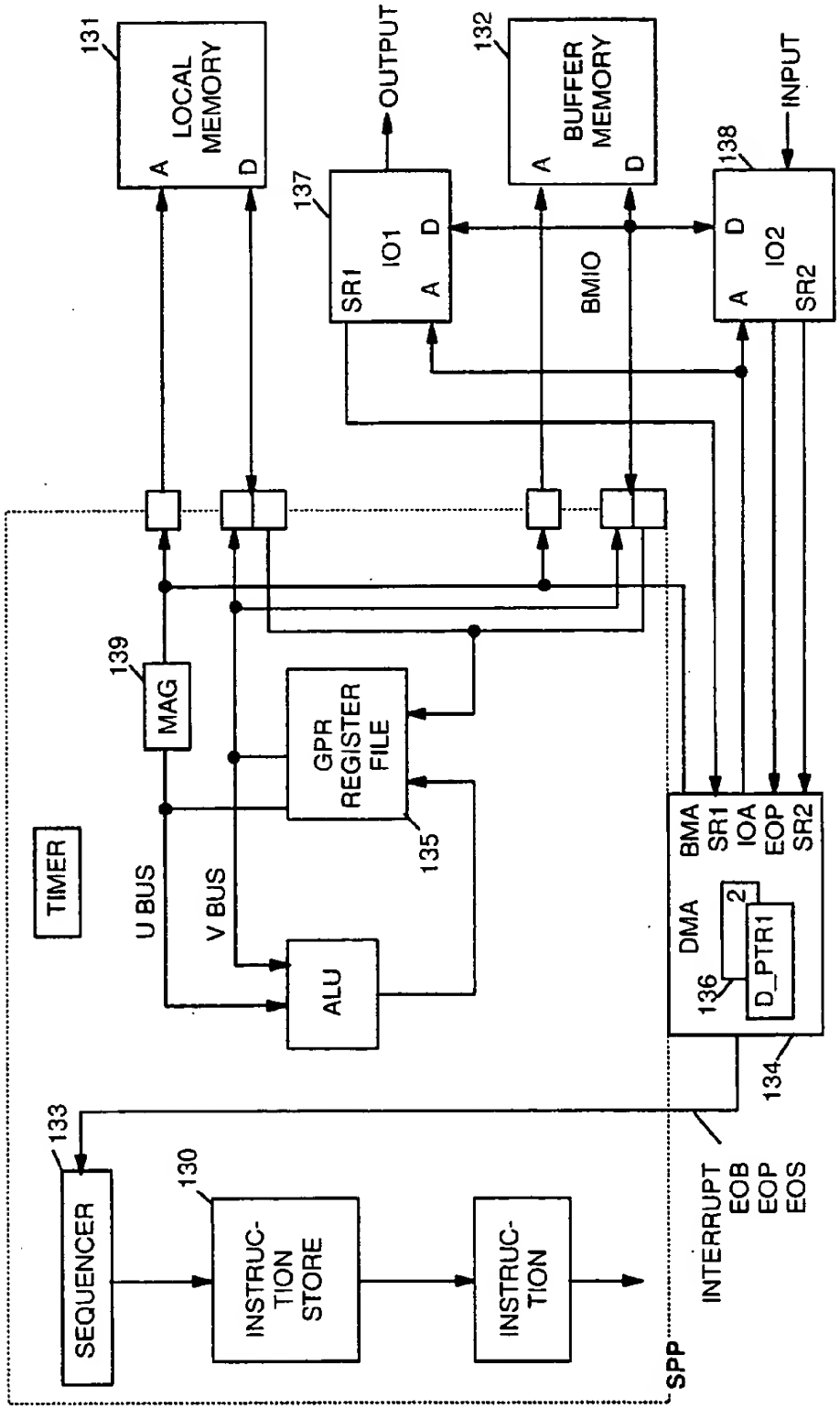

131
LOCAL MEMORY
A
D
OUTPUT
132
BUFFER MEMORY
A
D
INPUT
138
137
SR1
IO1
A
D
BMIO
D
IO2
A
SR2
139
MAG
GPR REGISTER FILE
135
TIMER
U BUS
V BUS
ALU
BMA
SR1
IOA
EOP
SR2
DMA
2
D_PTR1
136
134
133
130
SEQUENCER
INSTRUC-TION STORE
INSTRUC-TION
INTERRUPT
EOB
EOP
EOS
SPP

FIG 13

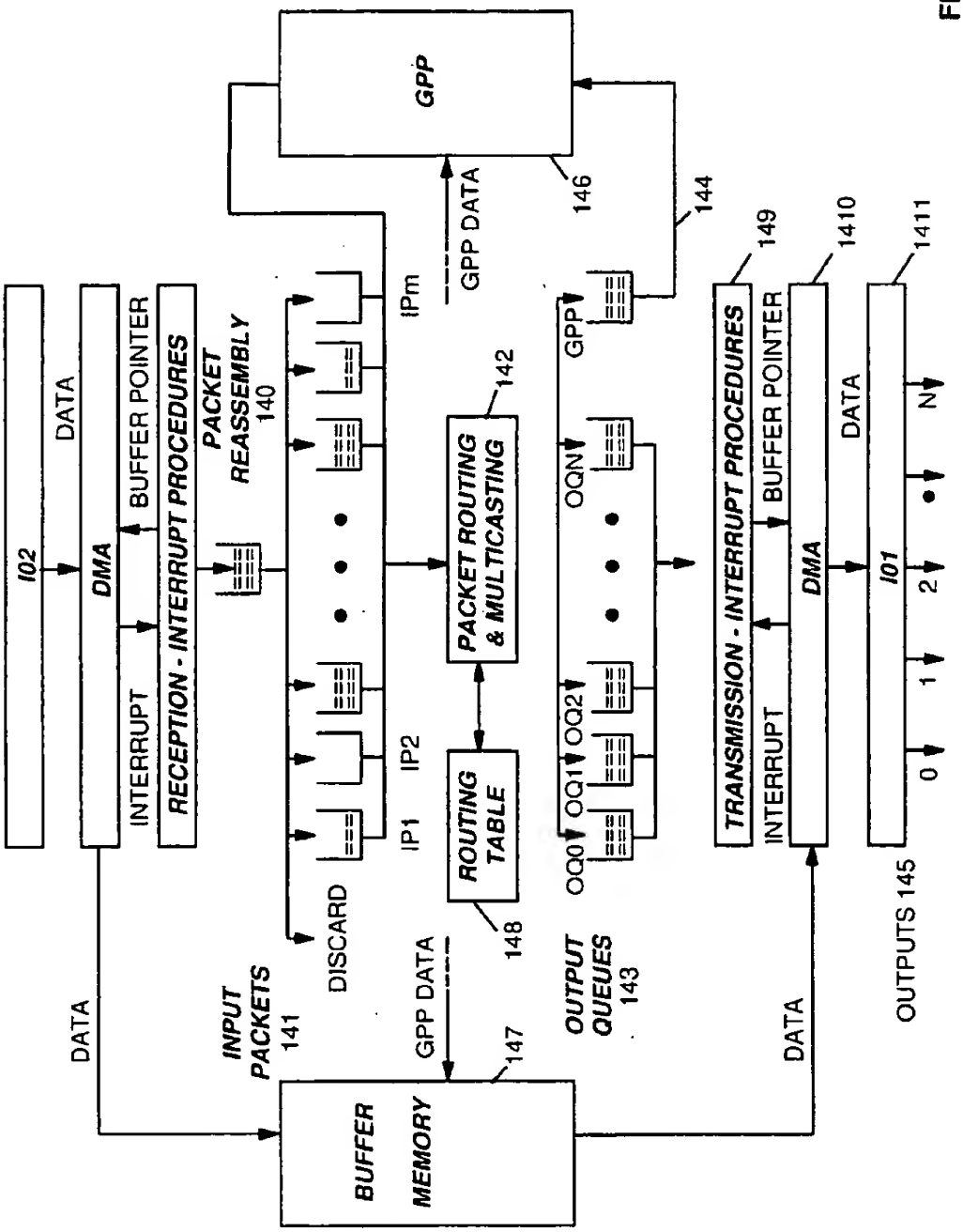

GPP
146
144
GPP DATA
IPm
I02
DATA
DMA
BUFFER POINTER
INTERRUPT
RECEPTION - INTERRUPT PROCEDURES
PACKET REASSEMBLY
140
142
PACKET ROUTING & MULTICASTING
ROUTING TABLE
148
IP1
IP2
DISCARD
INPUT PACKETS
141
GPP DATA
147
BUFFER MEMORY
DATA
OUTPUT QUEUES
143
OQ0
OQ1
OQ2
OQN
GPP
TRANSMISSION - INTERRUPT PROCEDURES
149
INTERRUPT
BUFFER POINTER
DMA
DATA
1410
I01
1411
0
1
2
N
OUTPUTS 145

FIG 14

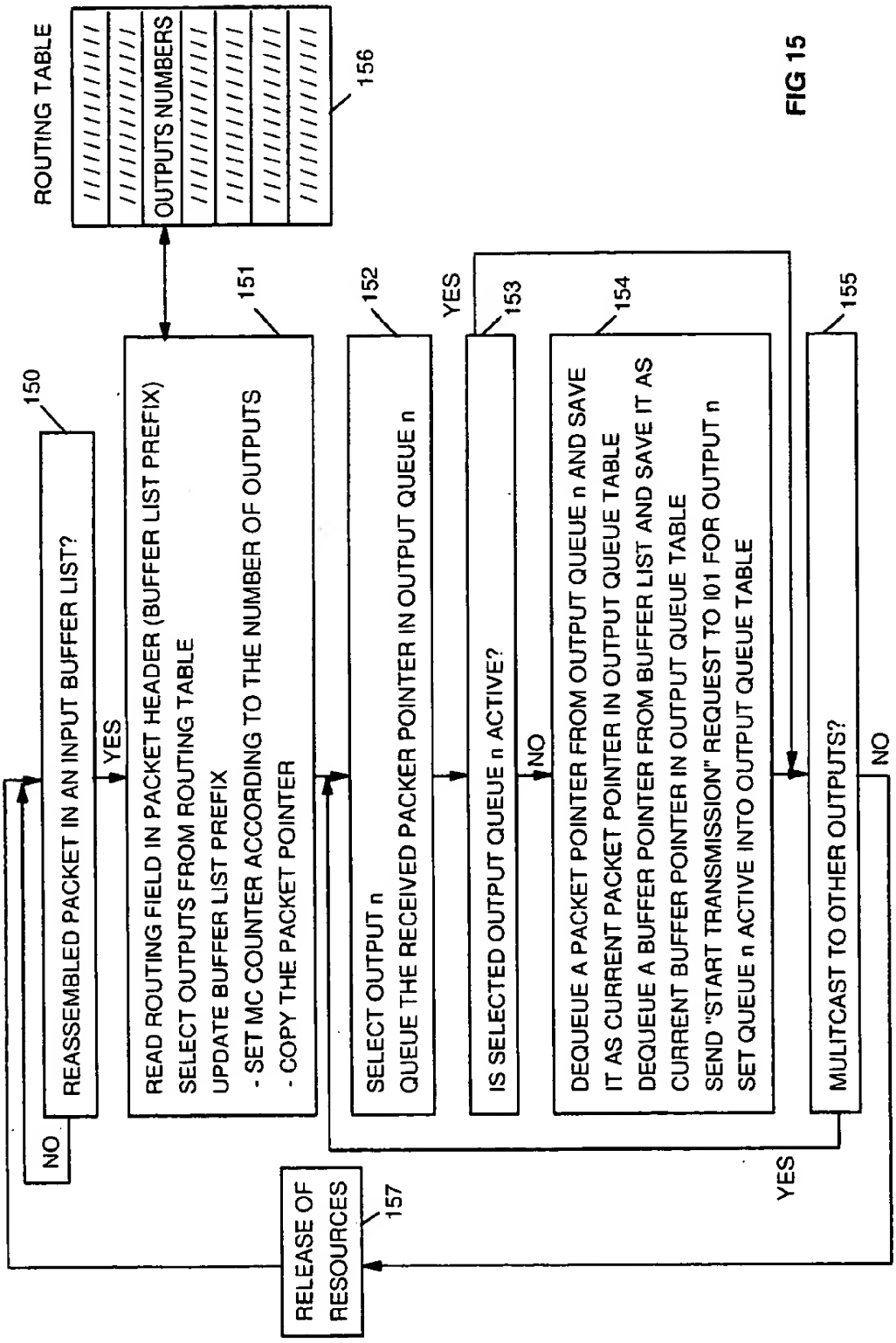

ROUTING TABLE
OUTPUTS NUMBERS
156
150
REASSEMBLED PACKET IN AN INPUT BUFFER LIST?
YES
NO
151
READ ROUTING FIELD IN PACKET HEADER (BUFFER LIST PREFIX)
SELECT OUTPUTS FROM ROUTING TABLE
UPDATE BUFFER LIST PREFIX
- SET MC COUNTER ACCORDING TO THE NUMBER OF OUTPUTS
- COPY THE PACKET POINTER
152
SELECT OUTPUT n
QUEUE THE RECEIVED PACKER POINTER IN OUTPUT QUEUE n
153
IS SELECTED OUTPUT QUEUE n ACTIVE?
YES
NO
154
DEQUEUE A PACKET POINTER FROM OUTPUT QUEUE n AND SAVE IT AS CURRENT PACKET POINTER IN OUTPUT QUEUE TABLE
DEQUEUE A BUFFER POINTER FROM BUFFER LIST AND SAVE IT AS CURRENT BUFFER POINTER IN OUTPUT QUEUE TABLE
SEND "START TRANSMISSION" REQUEST TO I01 FOR OUTPUT n
SET QUEUE n ACTIVE INTO OUTPUT QUEUE TABLE
155
MULITCAST TO OTHER OUTPUTS?
YES
NO
RELEASE OF RESOURCES
157

FIG 15

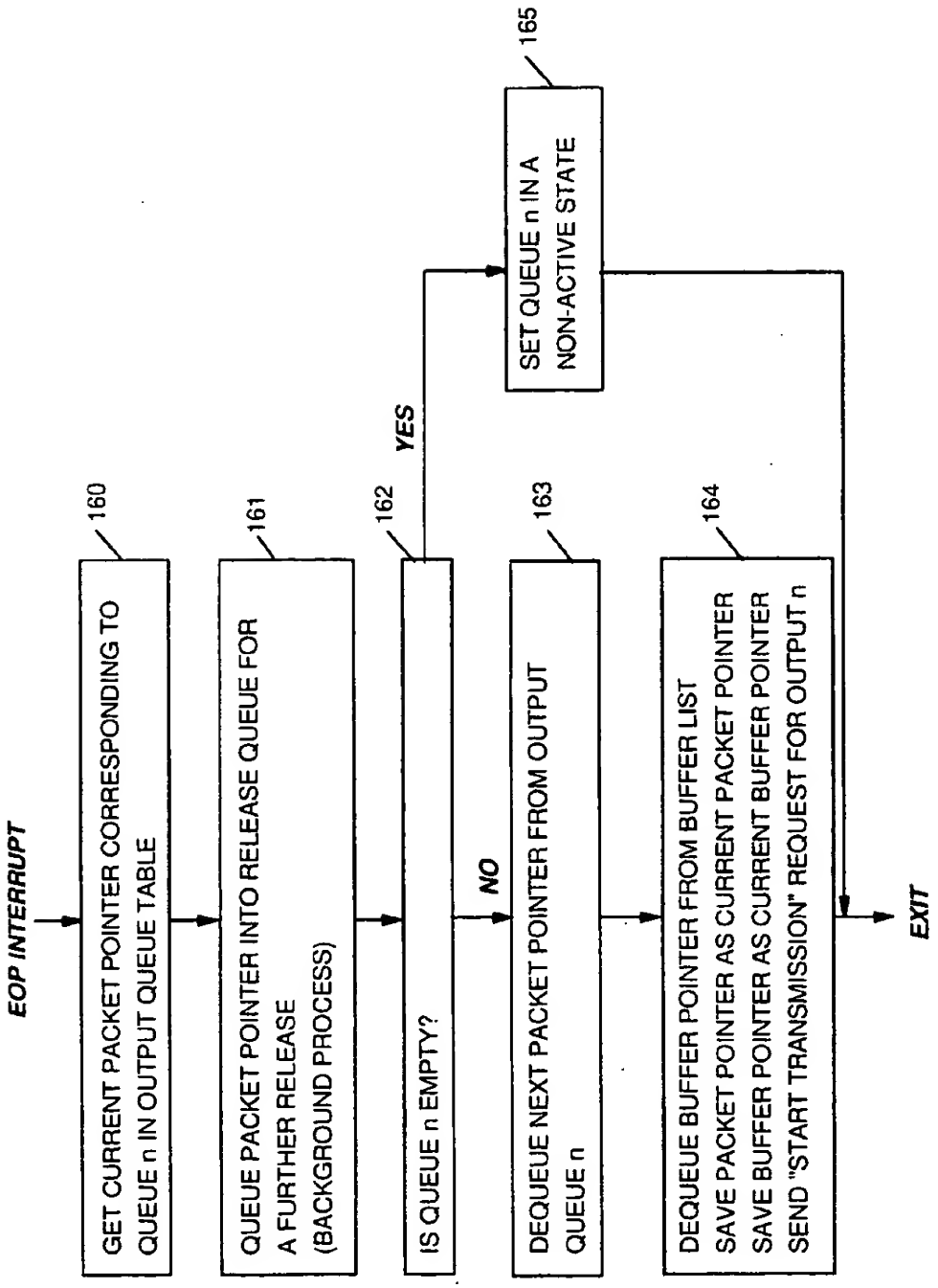
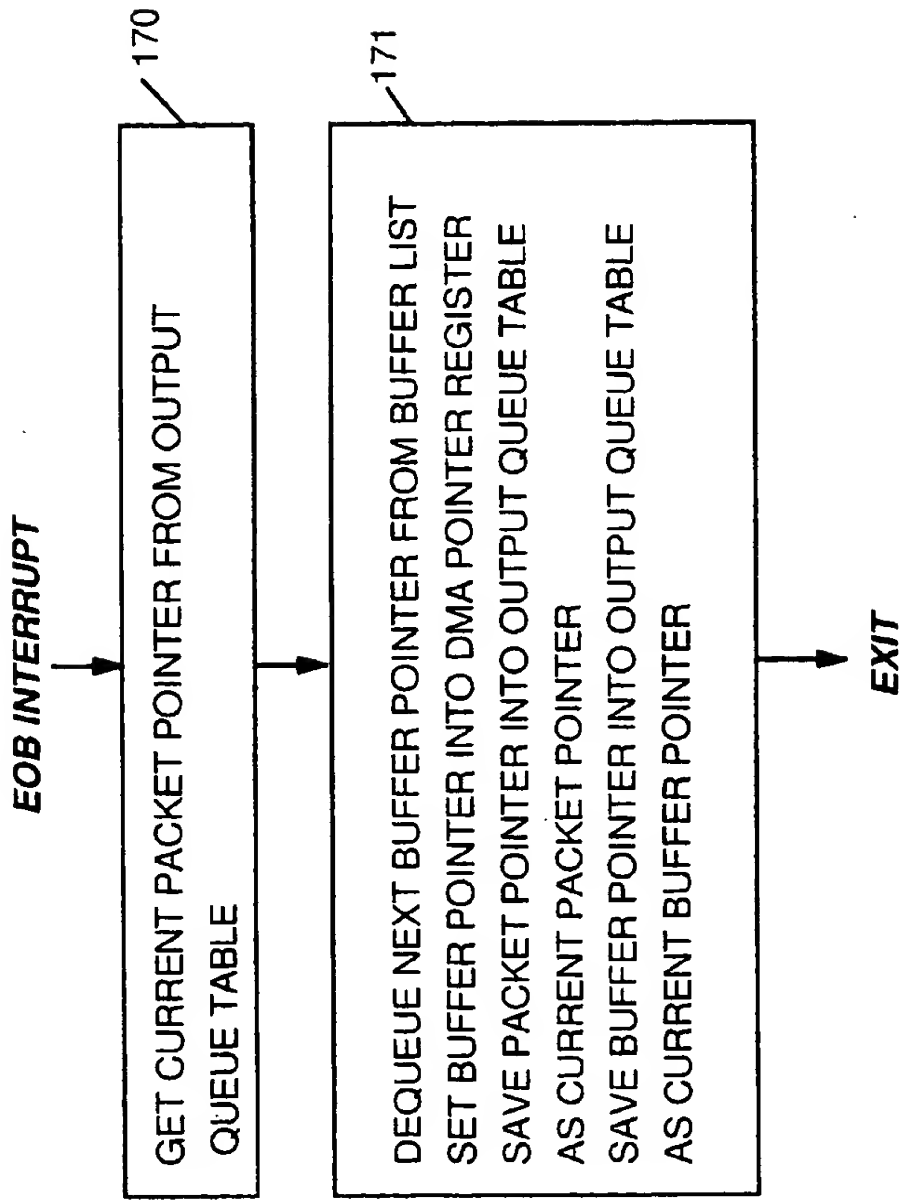

EOP INTERRUPT
160
GET CURRENT PACKET POINTER CORRESPONDING TO QUEUE n IN OUTPUT QUEUE TABLE
161
QUEUE PACKET POINTER INTO RELEASE QUEUE FOR A FURTHER RELEASE (BACKGROUND PROCESS)
162
IS QUEUE n EMPTY?
YES
NO
165
SET QUEUE n IN A NON-ACTIVE STATE
163
DEQUEUE NEXT PACKET POINTER FROM OUTPUT QUEUE n
164
DEQUEUE BUFFER POINTER FROM BUFFER LIST
SAVE PACKET POINTER AS CURRENT PACKET POINTER
SAVE BUFFER POINTER AS CURRENT BUFFER POINTER
SEND "START TRANSMISSION" REQUEST FOR OUTPUT n
EXIT

FIG 16

FIG 17


EOB INTERRUPT
170
GET CURRENT PACKET POINTER FROM OUTPUT QUEUE TABLE
171
DEQUEUE NEXT BUFFER POINTER FROM BUFFER LIST
SET BUFFER POINTER INTO DMA POINTER REGISTER
SAVE PACKET POINTER INTO OUTPUT QUEUE TABLE AS CURRENT PACKET POINTER
SAVE BUFFER POINTER INTO OUTPUT QUEUE TABLE AS CURRENT BUFFER POINTER
EXIT

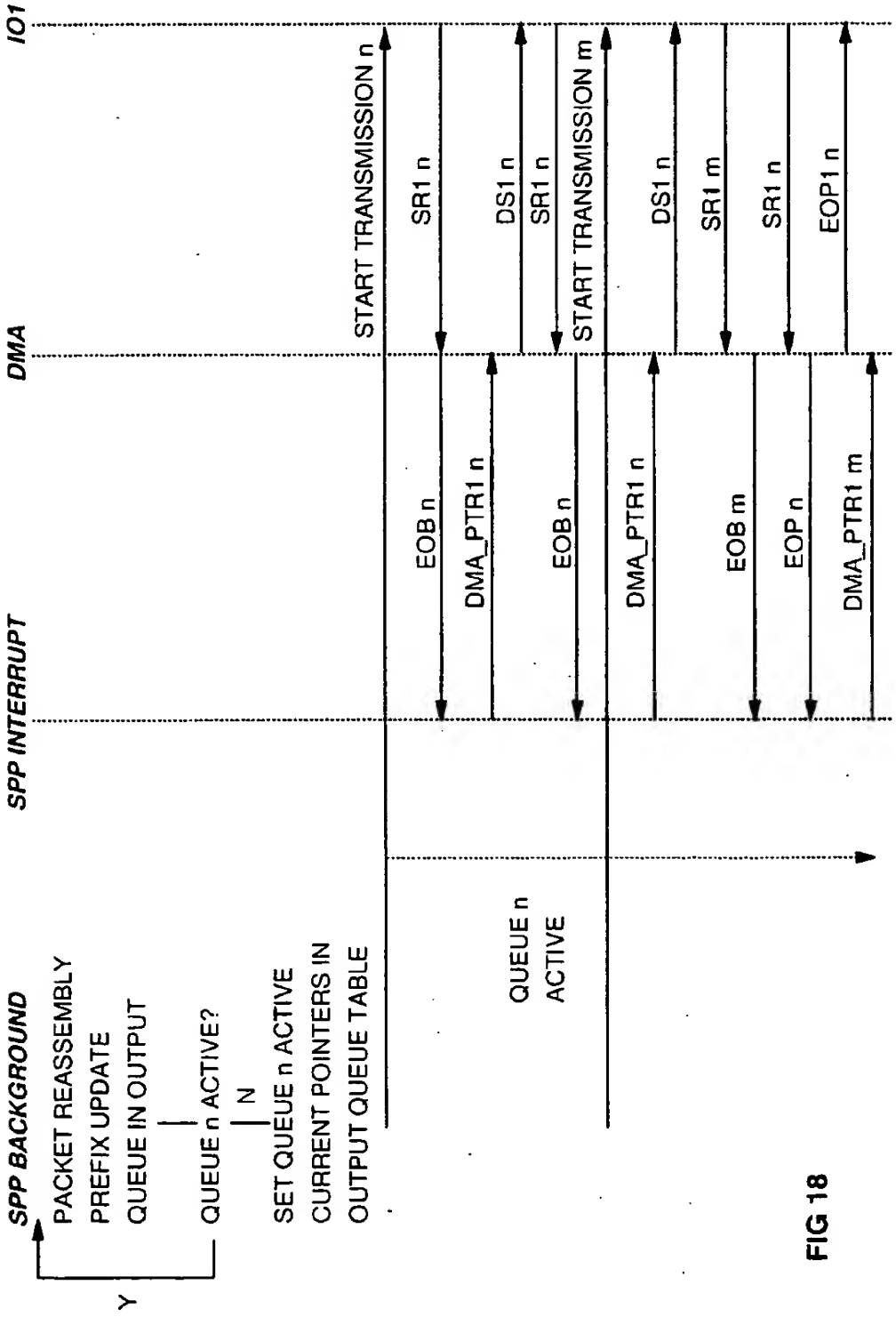

IO1
DMA
SPP INTERRUPT
SPP BACKGROUND
START TRANSMISSION n
SR1 n
DS1 n
SR1 n
START TRANSMISSION m
DS1 n
SR1 m
SR1 n
EOP1 n
EOB n
DMA_PTR1 n
EOB n
DMA_PTR1 n
EOB m
EOP n
DMA_PTR1 m
QUEUE n ACTIVE
PACKET REASSEMBLY
PREFIX UPDATE
QUEUE IN OUTPUT
QUEUE n ACTIVE?
N
SET QUEUE n ACTIVE
CURRENT POINTERS IN
OUTPUT QUEUE TABLE
Y

FIG 18

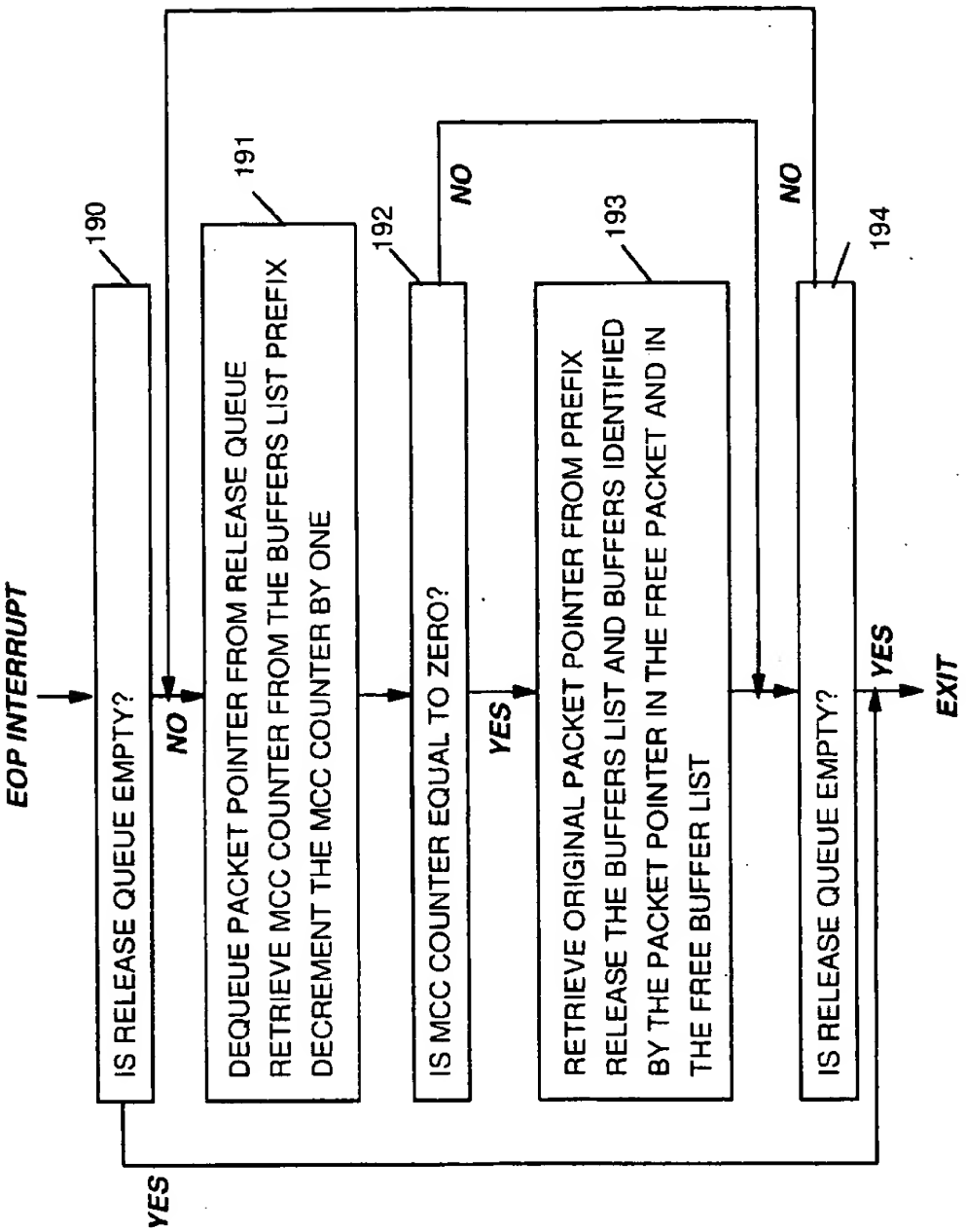

EOP INTERRUPT
190
IS RELEASE QUEUE EMPTY?
NO
191
DEQUEUE PACKET POINTER FROM RELEASE QUEUE
RETRIEVE MCC COUNTER FROM THE BUFFERS LIST PREFIX
DECREMENT THE MCC COUNTER BY ONE
192
IS MCC COUNTER EQUAL TO ZERO?
NO
YES
193
RETRIEVE ORIGINAL PACKET POINTER FROM PREFIX
RELEASE THE BUFFERS LIST AND BUFFERS IDENTIFIED
BY THE PACKET POINTER IN THE FREE PACKET AND IN
THE FREE BUFFER LIST
194
IS RELEASE QUEUE EMPTY?
NO
YES
EXIT
YES

FIG 19

EFFICIENT POINT-TO-POINT AND MULTI-POINT ROUTING MECHANISM FOR PROGRAMMABLE PACKET SWITCHING NODES IN HIGH SPEED DATA TRANSMISSION NETWORKS

TECHNICAL FIELD

The present invention relates to an efficient point-to-point and multi-points routing system and method for programmable data communication adapters in packet switching nodes of high speed networks.

BACKGROUND ART

The telecommunication environment is in full evolution and has changed considerably this recent years. The principal reason has been the spectacular progress realized in the communication technology:

the maturing of fiber optical transmission. High speed rates can now be sustained with very low bit error rates.

the universal use of digital technologies within private and public telecommunications networks.

In relation with these new emerging technologies, the offer of the telecommunication companies, public or private, are evolving:

The emergence of high speed transmissions entails an explosion in the high bandwidth connectivity.

the increase of the communication capacity generates more attractive tariffs.

A higher flexibility is offered to the users to manage their growth through a wide range of connectivity options, an efficient bandwidth management and the support of new media.

Once sampled and digitally encoded, voice, video and image derived data can be merged with pure data for a common and transparent transport.

Abundant, cheap communications means that many potential applications that where not possible before because of cost are now becoming attractive. In this environment, three generic requirements are expressed by the users:

Doing old applications better,

Optimizing communication networks,

Doing new applications.

High Performance Networks

In a first step, T1 backbone networks were primarily deployed with TDM (Time Division Multiplexing) technology to achieve cost savings through line aggregation. These systems easily supported the fixed bandwidth requirements of host/terminal computing and 64 Kbps PCM (Pulse Code Modulation) voice traffic.

The data transmission is now evolving with a specific focus on applications and by integrating a fundamental shift in the customer traffic profile. Driven by the growth of workstations, the local area networks (LAN) interconnection, the distributed processing between workstations and super computers, the new applications and the integration of various and often conflicting structures—hierarchical versus peer to peer, wide (WAN) versus local (LAN) area networks, voice versus data—the data profile has become higher in bandwidth, bursting, non deterministic and requires more connectivity. Based on the above, it is clear that there is strong requirement to support distributed computing applications across high speed backbones that may be carrying

LAN traffic, voice, video, and traffic among channel attached hosts, business workstations, engineering workstations, terminals, and small to intermediate file servers. This traffic reflects a heterogeneous mix of:

end user network protocols including Ethernet, Tocken Ring, APPN, FDDI, OSI, ISDN, ATM . . . , and

real time (steady stream traffic such as voice and video) and non real time (bursty nature traffic such as interactive data) transmissions.

This vision of a high speed protocol-agile backbone network is the driver for the emergence of fast packet switching networks architectures in which data, voice, and video information is digitally encoded, chopped into small packets and transmitted through a common set of nodes and links. Although low speed links may exist, the availability of fiber optic links will make cost effective to have a few links of high speed rather that many links of low speed. In addition to the high speed backbone, there exists a peripheral network which essentially provides access to the switching nodes. This peripheral network is composed of relatively low speed links which may not use the same protocols or switching techniques used in the backbone. In addition, the peripheral network performs the task of multiplexing the relatively slow end users traffic to the high speed backbone. Thus, backbone switching nodes are principally handling high speed lines. The number of high speed links entering each switching node is relatively small but the aggregate throughput very high in the Giga-bits per second range.

Throughput

The key requirement of these new architectures is to reduce the end-to-end delay in order to satisfy real time delivery constraints and to achieve the necessary high nodal throughput for the transport of voice and video. Increases in link speeds have not been matched by proportionate increases in the processing speeds of communication nodes and the fundamental challenge for high speed networks is to minimize the packet processing time within each node. As example, for meeting a typical 100 ms delay to deliver a voice packet between two end users:

A total of 36 ms might be needed for the packetization and play-out functions at the end points.

About 20 ms is the unalterable propagation delay needed, say, to cross the United States.

There remains 44 ms for all the intra-node processing time as the packet moves through the network. In a 5 nodes network, each node would have about 8 ms for all processing time including any queueing time. In a 10 nodes network, each node would have about 4 ms.

Another way of looking the same constraint is illustrated in FIG. 1: taking a node with an effective processing rate of 1 MIPS (Millions of Instructions Per Second), it is possible to fill a 9.6 kbps line with 1000 byte packets even if a network node must execute 833 000 instructions per packet processed. For a 64 kbps line the node can afford 125 000 instructions per packet. In order to fill an OC24 link, however, our 1 MIPS node could only execute 7 instruction per packet! In the latter case even an effective rate of 10–30 MIPS would allow only 70–200 instructions per packet.

In order to minimize the processing time and to take full advantage of the high speed/low error rate technologies, most of the transport functions provided by the new high bandwidth network architectures are performed on an end-to-end basis. This includes the flow control and error recov-

ery for data, the packetization and reassembly for voice and video. The protocol is simplified:

First, there is no need for transit node to be aware of individual (end user to end user) transport connections.

Secondly high performance and high quality links does not require any more node to node error recovery or retransmission. Congestion and and flow control are managed at the access and end points of the network connections reducing both the awareness and the function of the intermediate nodes.

Packet size

Transmission of real time data, like voice or video packets, which must be delivered to the receiver at a steady, uniform rate (isochronous mode) requires the use of short packets. In another side, pure data does not have any problem with transit delay. They are generated in a very bursty and non deterministic manner. The longer packet are, the fewer packets per second must be switched for a given data throughput. In order to take full advantage of the different data packet transmission systems, the data transfer across the network must be done with packets of nearly the same size as the user packets without processing them into artificial lengths. As opposed to solely data networks or solely voice or video networks, the high speed network architectures have to support a plurality of heterogeneous transmission protocols operating with variable length packets.

Connectivity

In a high speed network, the nodes must provide a total connectivity. This includes attachment of the customer's devices, regardless of vendor or protocol, and the ability to have the end user communicate with any other device. Traffic types include data, voice, video, fax, graphic, image. The node must be able to take advantage of all common carrier facilities and to be adaptable to a plurality of protocols: all needed conversions must be automatic and transparent to the end user. For example, a high speed node must not have any dependencies on the existence of SNA (System Network Architecture) equipments on a user network. It has to be able to offer a similar level of service in a SNA environment as in a non-SNA environment made of routers, Private Branch eXchanges (PBXs), Local Area Networks (LAN) . . . .

Key Requirements

The efficient transport of mixed traffic streams on very high speed lines means for each communication node of the network a set of requirements in term of performance and resource consumption which can be summarized as follows:

a very short packet processing time,

a very high throughput,

an efficient queue and buffer management,

a limited number of instructions per packet,

a minimum impact of the control flow on the user traffic, and

a very large flexibility to support a wide range of connectivity options.

The high bandwidth dictates the need of specialized hardware to support very fast packet handling and control protocols, and to satisfy the real time transmission needs of the voice and video traffic. The processing time being the main bottleneck in high speed networks, most of the communication nodes today are built around high speed switching hardware to off-load the routing packet handling and routing functions from the processor.

However, on equal performances, a software approach represents the most adequate solution for each node to meet the connectivity and flexibility requirements and to optimize the manufacturing and adaptation costs. The line adapters, are based on a common hardware design and are configured by means of a specific programming to execute either the access point or inter nodal transport functions. The adaptability of the adapters to support different access protocols and data streams—Frame Relay, HDLC (High level Data Link-Control), CBO (Continuous Bit Operations), ATM (Asynchronous Transfer Mode), . . .—is provided by logical components called Access Agents. Such logical associations Adapter/Access Agent are specified by software, providing a very large flexibility at a reduced cost. Each line adapter is automatically configured at system initiation according to:

the adapter function, and

the access protocol.

Programmable High Performance Communication Nodes

The throughput of a communication adapter is defined as the total time required to handle a data packet from the input to the output. However, the packet size being application dependent, two measures are currently used to evaluate the performances of adapters:

first, the number of packets of fixed length the adapter is able to handle in a second (packet throughput),

second, the number of bits per second the adapter is able to transmit in case of infinite packet length (data throughput).

The performances depend on the hardware and on the processor capabilities but the main throughput limiting factor is the packet processing time and this processing time is directly related to the number of instructions required to handle a packet. The operations on the packets can be divided in two categories:

the background process with the operations of packet routing, assembling—disassembling, formatting, bandwidth management, priority, ... This process is designed according to the adapter function, but in the same application, the background process is identical for all packets independently of their size.

the buffering process with the interrupt routines. This operation is generic for all adapter types. The processing time required for this operation is directly proportional to the packet length.

The interrupt routines are the way real time is supported by the processor. They must be as short as possible to avoid the overrun on the input device and the underrun on the output device. They commands the performance of the adapter in term data throughput (bits per second). For packets of infinite length, the background process disappears and the throughput bottleneck is the interrupt response time which depends of the queuing and dequeueing operations.

In another way, to maximize the packet throughput (number of packets per seconds that the adapter is able to transmit), the number of instructions required by the background process must be reduced to a minimum.

Non published European patent application 93480087.1 (IBM docket FR993027) entitled *Programmable High Per-*

formance Data Communication Adapter for Packet Transmission Networks (prior art under Article 54(3) EPC), which content is herein incorporated by simple reference, discloses a high performance packet buffering method and system in a programmable data communication adapter. The adapter includes a programmable processor, for receiving and transmitting data packets of fixed or variable length. The system is designed to optimize the queueing and dequeueing operations and in particular to minimize the number of instructions to manipulate the packets. It is characterized in that it comprises:

means for storing data packets in buffers,

means for identifying said data packets in the buffers

means for queueing in a local memory the packet identifiers in a single instruction,

means for dequeueing from the local memory the packet identifiers in another single instruction,

means for relcasing the buffers.

Each instruction comprises up to three operations executed in parallel by said processor:

an arithmetical and logical (ALU) operation on the packet identifiers,

a memory operation on the local memory, and

a sequence operation.

SUMMARY OF THE INVENTION

An efficient point-to-point and multi-points routing system and method for data communication adapters in packet switching nodes of a high speed network is disclosed. The general principles of the present invention can be summarized as follows:

data packets are never copied during their routing through the adapter, only packet pointers are copied for each destination,

each output is processed independently on a priority mode,

no overhead generated by the multi-points routing in the interrupt procedures,

release of the resources on a non priority mode (background procedure).

DESCRIPTION OF THE DRAWINGS

FIG. 1 shows the processing times (or number of instructions per second) required in function of the different line throughputs supported by the present invention.

FIG. 2 shows a typical model of high speed packet switching network including the access and transit nodes claimed in the present invention.

FIG. 3 describes a high speed Routing Point according to the present invention.

FIG. 4 shows a programmable high performance adapter according to the present invention.

FIG. 5 represents the receive and transmit data flows in a Trunk Adapter.

FIG. 6 illustrates the buffer, packet and queue structures according to the present invention.

FIG. 7 illustrates the Control Point Spanning Tree.

FIG. 8*a* shows a graphical representation of a typical header for packets transmitted in a network such as represented in FIG. 2.

FIG. 8*b* shows a graphical representation of a Multicast Tree Routing Field in the header represented in FIG. 8*a*.

FIG. 9 represents the List Pointer structure according to the present invention.

FIG. 10 shows an overview of the multicast mechanism as claimed in the present invention.

FIG. 11 represents the Buffer List structure of a packet supporting the multicasting routing according to the present invention.

FIG. 12 represents the Free Buffer List structure according to the present invention.

FIG. 13 represents the Processor functional structure according to the present invention.

FIG. 14 shows a general view of the packet processing in the programmable adapter according to the present invention.

FIG. 15 shows the packet routing and multicasting process according to the present invention.

FIG. 16 shows the interrupt procedure at packet level according to the present invention.

FIG. 17 shows the interrupt procedure at buffer level according to the present invention.

FIG. 18 illustrates the data flow between the Specific Purpose Processor (SPP), the Direct Access Memory (DMA) and IO1 device according to the present invention.

FIG. 19 is a flow chart illustrating the release mechanism of the memory resources in the background procedure as claimed in the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT OF THE INVENTION

As illustrated in FIG. 2, a typical model of communication system is made of several user networks (212) communicating through a high performance network (200) using private lines, carrier provided services, or public data networks. Each user network can be described as a set of communication processors and links (211) interconnecting large computers used as Enterprise Servers (213), user groups using workstations or personnel computers attached on LAN (Local Area Networks 214), applications servers (215), PBX (Private Branch eXchange 216) or video servers (217). These user networks, dispersed in different establishments, need to be interconnected through wide area transport facilities and different approaches can be used for organizing the data transfer. Some architectures involve the checking for data integrity at each network node, thus slowing down the transmission. Others are essentially looking for a high speed data transfer and to that end the transmission, routing and switching techniques within the nodes are optimized to process the flowing packets towards their final destination at the highest possible rate. The present invention belongs essentially to the latter category and more particularly to the fast packet switching network architecture detailed in the following paragraphs.

High Speed Packet Switching Networks

The general view in FIG. 2 shows a fast packet switching transmission system comprising eight nodes (201 to 208) each node being interconnected by means of high speed communication lines called Trunks (209). The access (210) to the high speed network by the users is realized through Access Nodes (202 to 205) located at the periphery. These Access Nodes comprise one or more Ports, each one pro-

viding an access point for attaching external devices supporting standard interfaces to the network and performing the conversions required to transport the users data flow across the network from and to other external devices. As example, the Access Node 202 interfaces respectively a Private Branch eXchange (PBX), an application server and a hub through three Ports and communicates through the network by means of the adjacent Transit Nodes 201, 208 and 205.

Switching Nodes

Each network node (201 to 208) includes a Routing Point where the incoming data packets are selectively routed on the outgoing Trunks towards the neighboring Transit Nodes. Such routing decisions are made according to the information contained in the header of the data packets. In addition to the basic packet routing function, the network nodes also provide ancillary services such as:

the determination of routing paths for packets originated in the node,

directory services like retrieving and updating information about network users and resources,

the maintaining of a consistent view of the physical network topology, including link utilization information, and

the reservation of resources at access points of the network.

Each Port is connected to a plurality of user processing equipments, each user equipment comprising either a source of digital data to be transmitted to another user system, or a data sink for consuming digital data received from another user system, or, typically, both. The interpretation of the users protocols, the translation of the users data into packets formatted appropriately for their transmission on the packet network (200) and the generation of a header to route these packets are executed by an Access Agent running in the Port. This network layer header (800) shown in FIG. 8*a* is made of Control (801), Routing (802) and Redundancy Check (803) Fields.

The Control Fields (801) include, among other things, an encoded identification of the protocol to be used in interpreting the Routing Field.

The Routing Fields (802) contain all the information necessary to route the packet through the network (200) to the destination End Node to which it is addressed. These fields can take several formats depending on the routing mode specified

The Redundancy Check Fields (803) are used to check for errors in the header itself. If an error is detected, the packet is discarded.

Routing Points

FIG. 3 shows a general block diagram of a typical Routing Point (300) such as it can be found in the network Nodes (201 to 208) illustrated in FIG. 2. A Routing Point comprises a high speed packet Switch (302) onto which packets arriving at the Routing Point are entered. Such packets are received:

from other nodes over high speed transmission links (303) via Trunk Adapters (304).

from users via application adapters called Ports (301).

Using information in the packet header, the adapters (304, 301) determine which packets are to be routed by means of the Switch (302) towards a local user network (307) or towards a transmission link (303) leaving the Node. The adapters (301 and 304) include queuing circuits for queuing packets prior to or subsequent to their launch on the Switch (302).

The Route Controller (305) calculates the optimum routes through the network (200) so as to minimize the amount of network resources used to complete a communication path and builds the header of the packets generated in the Routing Point. The optimization criteria includes the characteristics of the connection request, the capabilities and the utilization of the Trunks in the path, the number of intermediate nodes .... All the information necessary for the routing, about the nodes and transmission links connected to the nodes, are contained in a Network Topology Database (306). Under steady state conditions, every Routing Point has the same view of the network. The network topology information is updated when new links are activated or new nodes added to the network. Such information is exchanged by means of control messages with all other Route Controllers to provide the necessary up-to-date information needed for route calculation (such database updates are carried on packets very similar to the data packets between end users of the network). The fact that the network topology is kept current in every node through continuous updates allows dynamic network reconfigurations without disrupting end users logical sessions

The incoming transmission links to the packet Routing Point may comprise links from external devices in the local user networks (210) or links (Trunks) from adjacent network nodes (209). In any case, the Routing Point operates in the same manner to receive each data packet and forward it on to another Routing Point as dictated by the information in the packet header. The fast packet switching network operates to enable a communication between any two end user applications without dedicating any transmission or node facilities to that communication path except for the duration of a single packet. In this way, the utilization of the communication facilities of the packet network is optimized to carry significantly more traffic than would be possible with dedicated transmission links for each communication path.

Control Point Spanning Tree

Associated with networks are several network control functions: network Spanning Tree maintenance, Topology Database, Directory, Path Selection, Bandwidth Management and Reservation, and Congestion Control. Preferably, every node has a set of the foregoing network control functions called its Control Point (CP) that the node uses to facilitate the establishment of connections between user applications.

The main purpose of the Control Point Spanning Tree is to ensure a communication and distribution mechanism for all network control functions in the nodes of a high speed network. It (logically) joins together the Control Points (305) if the nodes are in a (physically) connected portion of the network. As illustrated in FIG. 7 a tree is a pattern of connections with no loop, the term "spanning" means that the tree spans (connects) all of the nodes. Once formed, the Control Point Spanning Tree is the principal system used to disseminate control information such as Topology Database (306) updates. This mechanism is fundamental to minimize delays due to intermediate node processing.

First, an intermediate node will get each control message exactly once on the tree, and

second, the message can be forwarded along outgoing links of the tree before the intermediate node has even looked at the packet contents.

A distributed algorithm creates and maintains the Control Point Spanning Tree in presence of node and link failures and helps to minimize the impact of the increased control flows that result when the network grows.

Routing Modes

The routing within the network presents two aspects:

1. Determining what the route for a given connection shall be.

2. Actually switching the packet within a switching node.

There are many methods of determining a route through a network. For very high throughput, once the route selected, the critical item is that the switching elements must be able to route an incoming packet in a very short portion of time. Driven by the requirements to keep transit node processing at a minimum, the transport services are designed to operate on an end-to-end basis so there is no hop-by-hop error recovery or retransmission envisioned for high speed, high performance (low error) links. There is also no need for transit nodes to be aware of individual transport connections.

Data packets are routed and queued in the transit nodes according to the routing information contained in the header. Several routing modes can be used in high speed networks (refer to an *Introductory Survey* (pages 116 to 129)—GG24-3816-01 ITSC Raleigh June 1993). However, in most of the case, packets using different modes can share the same data transmission facilities.

To classify the different transport type, we can consider on one side the point-to-point transmissions on the other side the multi-pointss transmissions. As illustrated by the following examples, each routing mode has its particular indented use and includes advantages and disadvantages that complement the other modes.:

Point-to-Point Transmission

Source Routing

The Source Routing is a particular implementation of the distributed routing for connectionless networks. The The source node (or access node) is responsible for calculating the route the packet must take through the network.

Each packet includes in its routing field a list of the labels of all links through which the packet will pass as it moves across the network. The Source Routing requires no connection setup activity in intermediate nodes and supports true datagram services.

Label Swapping

This routing mode is used in connection oriented networks. Each packet sent on the link has a header which includes an arbitrary number identifying which logical connection that this packet belongs to. Label Swapping requires that connection tables be set up and maintained dynamically in each node. Due to the low packet overhead, this technique is particularly adapted to the transmission of very short packets (for example real-time voice connections).

Multi-Points Transmission

Multicast allows one entity to communicate with multiple entities. An efficient multicast mechanism provides packet delivery to a set of users without having to broadcast to all users in the network or having to "unicast" separate copies of packets to each user of a group.

Multicast Tree Routing

Multicast Tree Routing is supported by the ability of each node to recognize that a particular label represents a predefined tree and to forward the received packet along all the outbound links associated with that tree. This is the basic routing mechanism that underlies both the fundamental Control Point Spanning Tree and any individual trees that can be established, pruned, and removed dynamically in a network. The Control Point Spanning Tree greatly reduces the negative effect on performance that network control flows could have in a high speed network. Tree routing on other Multicast Trees can be fundamental to supporting, for example, multi-attached LANs, or specific applications like video conference.

Remote Access to a Multicast Tree

Remote Access to a Broadcast Tree is a straightforward combination of the two routing modes: Source Routing and Multicast Tree Routing. The routing field includes several Source Routing labels followed by a tree address label. This allows a node that is not a member of a tree to route packets to nodes that are tree members. This technique is described with more details in European patent application 93480059.0 filed May 19th 1993 entitled *Method and Apparatus for Routing Packets in Packet Transmission Networks.*

Multicast Mechanism

The Multicast Tree Routing provides a broadcast capability over a preselected tree that can connect multiple nodes. Any member of a multicast group can send a packet which will be forwarded to all other member of the group. A Multicast Tree is simply identified, in the Routing Point, by a reference in the Topology Database. Different trees can coexist in the same network. One internal use of a Multicast Tree is to join all the Control Points of a network (Control Point Spanning Tree) for maintaining in each Topology Database a true image of the network configuration. This broadcast mechanism can also be used to address subset of network users (like a closer user group).

As illustrated in FIG. 8*b*, a Multicast Tree is identified by a tree address (**804**) which is part of the routing field (**802**) of packets to be sent on the tree. Unlike the Source Routing labels which are uniquely preassigned to individual links within each node, the tree address is assigned to all links that are to be part of a tree at the time that the tree is set up. The tree address must be unique to one and only one tree across all nodes over which the tree passes. When a packet addressed to a Multicast Tree arrives at a node over a link, any links in that node that are configured for that tree address will forward this packet. In this manner, copies of the packet flow across the network, along each link configured as a branch of the tree, with one and only one copy arriving at each node that is part of the tree. A Hop Countdown Field (**805**) is included in the Multicast Tree Routing Field (**802**) which is decremented at each retransmission of the packet. When the hop countdown is equal to zero, no further retransmission is permitted and an error condition is assumed. The routing field is terminated by and End Of Field (EOF) flag (**806**).

Any node can be simultaneously be a member of a plurality of different trees and hence the tree addresses assigned to overlapping multicast trees must be unique. Nodes can be added or deleted from a Multicast Tree simply by adding or removing the tree address from the links leading to the node to be added or deleted. A more detailed description is disclosed in European patent application 93480060.8 filed May 19th 1993 entitled *Multicast Com-*

munication Tree Creation and Control Method and Apparatus.

Ports and Trunk Adapters

Adapters Function

Ports are located at the boundary of the high speed network. They allow terminal equipments to exchange information through the high speed network without the need for knowing the specific high speed protocol used. The main function of the Ports are:

receiving foreign protocol data units from an external resource and forwarding them as high speed packets over the network to a target Port, and

converting high speed packets back to foreign protocol data units and sending them to the target resource,

controlling the bandwidth.

Note: the source and target Ports may be located in the same node.

Trunks are the links between the high speed network nodes. They carry high speed packets. Each Trunk manage its link bandwidth and link status. A critical task of the Trunks is the management of traffic priorities and allocation of internal buffers to reduce delay and congestion.

In addition, there is a special type of adapter called Route Controller Adapter (305) which:

communicates with the other adapters (301, 304) through the Switch (302),

implements the centralized functions of the Route Controller such as the topology, the path selection . . . ,

establishes, maintains and cancels end-to-end high speed connections.

Adapters Architecture

Several techniques for designing said Ports, Trunk and Route Controller Adapters, to obtain more or less flexible and efficient transmission systems. Most of the adapters today are built around a specialized hardware depending on the function and protocol of the connected links.

The present invention, to satisfy the previously enumerated connectivity and flexibility requirements, provides a software solution based on a common hardware structure. Port and Trunk Adapters present the same architecture and their functional differentiation is realized through a specific programming. However, even using the most efficient general purpose microprocessor today available on the market, the experience shows that it is very difficult to reach the desired level of performance in term of number of switched packet per second. This is the reason why the control of each adapter has been shared between two processors: a Specific Purpose Processors (SPP, 406, 412), optimized for the packet switching and a General Purpose Processor (GPP, 409), the first dealing with the packet to be switched, the critical processing in term of performance, and the second with the adapter management.

As shown in FIG. 4, each adapter (400) comprises the following logic components:

1. A General Purpose Processor (GPP, 409) whose programming depends of the selected Port or Trunk Adapter function. The GPP implements the adapter control operations.

2. A Receive Adapter (401) for implementing three functions:

the checking of the high speed packets header.

the traffic discrimination according to the routing mode specified in the header of every incoming packet,

the routing of the incoming packets through the Switch (403) with the appropriate header.

The Receive Adapter includes:

a. a Line Receiver (407) for handling the data movements between a Line Interface (415) and a Receive Buffer Memory (RBM, 405).

b. a Receive Buffer Memory (RBM, 405) to temporarily store users data packets.

c. a Receive Specific Purpose Processor (RSPP, 406) based on a specialized microprocessor comprising a Local Memory (LM, 408). The RSPP handles the received steady state packet flow and forwards the control packets to the General Purpose Processor (409).

d. a Local Memory (LM, 408) used by the RSPP (406) as work space.

e. a Switch Transmitter Adapter (404) for

handling the data flow transferred from the buffer Memory (RBM, 405) under the control of the Receive Specific Purpose Processor (406)

segmenting this flow in fixed length cells and,

generating an appropriate switch routing header

3. a Transmit Adapter (402) for implementing the following functions:

the reception of the data flow from the Switch (403),

the checking of the cells header.

the reassembly in packets (Port),

the Trunk functions (Trunk adapter),

the routing.

The Transmit Adapter includes:

a. a Switch Receiver (410) for handling the flow coming from the Switch (403) and transferring it to the Buffer Memory for reassembly.

b. a Transmit Specific Purpose Processor (XSPP, 412) similar to the Receive Specific Purpose Processor (406). The XSPP handles the steady state data and forwards the control flow to the General Purpose Processor (GPP, 409).

c. a Line Transmitter Adapter (413) for handling the data movements between the Buffer Memory (411) and the Line Interface (415).

The adapters are connected on one side on the packet Switch and on the other side on the Line Interfaces:

The Line Interfaces (415) are used to adapt the Port and Trunk Adapter physical interfaces to the appropriate media.

The packet Switch (302, 403) allows the Route Controller (305) and the different Ports (301), Trunk Adapters (304) to communicate.

Data Flow Control

The receive and transmit data flows in the Trunk Adapters are represented in FIG. 5. In a proprietary high speed network with packets of variable lengths, the receive process involves the steps of:

1. Line Receiver, Buffer Memory, Specific Purpose Processor (501) system

a. receiving the packets from the line,

b. checking the packets header and in case of error discarding the packets,

c. processing the information contained in the packets header according the routing mode,

d. routing the control messages towards the General Purpose Processor (GPP, 502)

e. encapsulating the packets with a specific switch header in function of the destination adapter,

f. forwarding the packets and the GPP (502) control messages to the Switch Transmitter Adapter (504),

2. Switch Transmitter Adapter (504)

a. segmenting the packets in cells of fixed length adapted to the Switch (503),

b. generating an error check field to ensure the integrity, of the switch header during the transmission of the cells over the Switch (503).

The transmit process, as for it, comprises the steps of:

1. Switch Receiver Adapter (505)

a. receiving the cells from the Switch (503),

b. checking the switch header and, in case of error, discarding the cell,

2. Line Receiver, Buffer Memory, Specific Purpose Processor (506) system

a. reassembling the data packets,

b. forwarding the control packets to the General Purpose Processor (502),

c. encapsulating the packets with a routing header,

d. receiving control packets from the GPP (502),

e. queueing data and control packets in the appropriate queues,

f. handling the outgoing packets with priority given to real time data (and then to non real time data).

It is possible to design the adapters to work either in a a proprietary environment with packets of variable length, or in a standard mode such as ATM (Asynchronous Transmission Mode) with short cells of fixed length, or in both where appropriate. In this last case, for performance purpose, cells routed on the switch are identical or similar to these defined in the ATM protocol with as result:

the elimination of the packets segmentation and reassembly steps in the Switch Transmitter (508) and Receiver Adapters (509),

a simplification of the switch header processing in the Specific Purpose Processor (507, 510).

Adapter Functional Structure

The present invention deals with the relationships between the Line Receiver/Transmitter, the Buffer Memory, the Specific Purpose Processor, and the Switch Adapter and in particular with the handling of the data flow in a way to optimize the throughput and the processing time inside the adapters. More specifically, the invention relates to a very high performance system for queuing, dequeueing and distributing data packets on external links.

The communication adapters are based on the following principles:

the Specific Purpose Processor is designed to minimize the number of operations necessary to manage the steady state data flow.

data packets and control data are managed separately in two distinct memories respectively the Buffer Memory and the Local Memory.

the data packets buffering, the queueing, dequeuing, the routing and multicasting mechanisms are identical for all Ports—Trunk Receive and Transmit adapters.

According to these considerations, the following conventions will be used to describe the invention:

The device which reads in the Buffer Memory and the device which writes into the Buffer Memory are designated respectively by IO1 and IO2. That means:

in the receive side of the adapter, IO1=Switch Transmitter and IO2=Line Receiver

in the transmit side of the adapter, IO1=Line Transmitter and IO2=Switch Receiver.

In the same way:

an input data stream goes from the switch or external line (IO2) to the Buffer Memory.

an output data stream goes from the Buffer Memory to the switch or external line (IO1).

Furthermore:

The meaning of "packet" is application dependent. It may be applied, for example, to an SDLC-frame from a Port, to a proprietary packet format from a Trunk, or to a cell received from an ATM Trunk. The term "packet" that will be used in the following paragraphs will not refer to a precise data unit format.

Data Structures

Buffers, Packets, Queues Structures

Data packets are stored in the Buffer Memory (BM) while control data are managed directly by the Specific Purpose Processor (SPP) in the Local Memory (LM). The basic unit of memory that can be allocated to an input (Line/Switch Receiver (IO2)) or output device (Line/Switch Transmitter (IO1)) is a buffer of fixed length. As illustrated in FIG. 6, each of these buffers is represented in the Local Memory by a pointer called Buffer Pointer (B_PTR). A pointer is a generic term to identify a logical data structure (buffer, packet, queue . . .) stored in the Buffer Memory. The storage of a packet requires one or several buffers. These buffers are stacked together using a list of pointers (B_LIST) which is itself represented by a Packet Pointer (P_PTR). A list of Packet Pointers (P_LIST), identified by a Queue Pointer (Q_PTR), designates a queue of several packets.

List Prefix

Each list, representing a specific packet or a queue structure, is preceded by a Prefix used for storing any type of information related to the data the structure contains. In Buffer Lists, the Prefix contains information related to the routing of the packet:

the packet header

the date of the packet reception

the packet length

All the processor operations on the packet header are realized in the the Local Memory (LM) without having to access to the data stored in the Buffer Memory (BM). Furthermore, when the processor (SPP) is working on the Local Memory, the DMA operations on the Buffer Memory are not disrupted. The result is a more efficient routing process and memory management. As illustrated in FIG. 11, the support of the multicasting mechanism is realized by means of specific fields in the Buffer List Prefix of a each packet:

a Multicast Counter (MCC)

The counter is initialized according to the number of outputs. The intitial counter value is one or superior to one depending on if the packet is intended to be transmitted to a single (MCC=1: Point-to-point Routing Mode) or to multiple destinations (MCC>1: Multi-points Routing

Mode). The counter value is decremented each time the packet to multicast is transmitted to an output. When the counter value is equal to zero, the routing process is terminated and the resources attached to the packet can be released.

a copy of the Packet Pointer (P_PTR)

The original Packet Pointer, with the initial HEAD (identification of the first Buffer Pointer in the list) and TAIL (identification of the next Buffer Pointer in the list) values, is saved in the Buffer List Prefix to allow, at the end of the transmission, the complete release of all Buffers attached to the packet.

Packet Segmentation

To facilitate the memory management, the lists used for packets and queues are of fixed length. Packets which are bigger than the buffer list can contain, are segmented. This method allows the lists (B_LIST) not to be sized at the maximum packet length.

Buffer Pointers

Buffers need not to be full and the data may start and end at any place. For example, it is possible to reserve the first bytes of a buffer to include a header a posteriori. A Status Field (SF) is used in the last Buffer Pointer of a list to designate an End Of Packet (EOP). The general format of a Buffer Pointers is described in European patent application 93480087.1 (IBM docket FR993027) entitled *Programmable High Performance Data Communication Adapter for Packet Transmission Networks*

List Pointers

Referring to FIG. 9, the List Pointer format consists of three fields:

the List IDentifier (LID): identification of the list,

the HEAD: identification of the first pointer of the list,

the TAIL: identification of the next pointer to attach to the list.

The queuing process comprises the steps of:

testing if list is full (if INCREMENTED(TAIL)=HEAD).

storing the pointer at the address identified by the TAIL field in the pointer list identified by the LID field,

incrementing the TAIL field of the List Pointer,

The dequeueing process comprises the steps of:

testing if the list is not empty (if HEAD not equal to TAIL).

reading the pointer at the address identified by the HEAD field in the pointer list identified by the LID field,

incrementing the HEAD field of the List Pointer when list is not empty.

This queueing and dequeueing method is described with more details in European patent application 93480087.1 (IBM docket FR993027).

Free Buffer List (FBL)

The management of the Buffer Memory is realized by means of a specific list called Free Buffer List (FBL). The FBL comprises the totality of the Buffer Pointers and its role is to provide a status of the memory occupation (FIG. 12) using the HEAD and TAIL fields of the Free Buffer List Pointer (P_FBL):

T: total number of buffers in the Buffer Memory.

HEAD: identification of the first free buffer of the list. Each time a new buffer is filled, the HEAD field is incremented.

TAIL: identification of the next free buffer of the list. Each time a new buffer is released, the TAIL field is incremented.

HEAD=TAIL: the Buffer Memory is full.

Incremented TAIL=HEAD: the Buffer Memory is empty.

The Free Buffer List (FBL) is created at initiation time and is permanent contrary to other lists which are created dynamically (Buffer, Packet or Queue Lists).

Note: in general, when a lack of resources is detected (Free Buffer List empty), then the packet which cannot be stored in the Buffer Memory is discarded.

Free Packet List (FPL)

Buffer Lists are of fixed length (fixed number of elements). The management of these lists in the Local Memory (LM) is realized by means of a specific list called Free Packet List (FPL). The FPL is created at initiation time and is permanent like the Free Buffer List. It comprises all the Buffer List structures that will be dynamically created or released during the packet receive and transmission process.

It is important to note that, in the present invention, pointers (Buffer, Packet or Queue pointers) are simply stacked in predefined lists (FIG. 6). The pointers are not chained that means that pointers do not contain the address of the next pointer in the list and they are not aware of the location of this next pointer. This list structure is designed to optimize in term of performance the buffer manipulation in very high speed adapters but not the memory resources.

Specific Purpose Processor Structure

The Specific Purpose Processor functional structure is illustrated in FIG. 13.

Processor Parallel Processing

The Specific Purpose Processor is designed to execute up to three operations in parallel:

1. ALU (Arithmetical and Logical Unit) operations on registers

2. Memory operations

3. Sequence operations

The parallelism requires to distinguish instructions from operations:

Instruction: it is the content of the code word. In term of assembler language, the instruction corresponds to one line of code. All instructions are executed in one processor cycle

Operation: an instruction may be composed of one or more operations which are executed simultaneously.

Memory Space

The SPP memory space is divided in three blocks:

the Instruction Store (130),

The Local Memory (LM, 131), which is the code working space,

The Buffer Memory (BM, 132) which is the repository for data packets when they pass through the adapter.

They all operate in parallel, the Instruction Store (130) under control of the Sequencer (133), the Local Memory under control of the processor code and the Buffer Memory (132) under the control of the Direct Access Memory (DMA, 134)).

Registers

The registers are divided in two categories:

1. the General Purpose Registers (GPR)

These registers are located in the Register File (RF, 135) and are available as instruction operands.

2. the Control Registers (CR)

The CR's are hardware registers which are used in specific functions and are available also as instruction operands. However, there are less degree of freedom in their use, as compared with the GPR's. In particular, two of these control registers (136) are located in the Direct Access Memory (DMA, 134).

CR1=D_PTR1 (DMA Pointer IO1)

CR2=D_PTR2 (DMA Pointer IO2)

DMA Pointers 1 and 2 are associated to input/output IO1 (137) and IO2 (138) and they both contain the current Buffer Pointer (B_PTR).

Memory Address Generator (MAG, 139)

In all load or store operations, on the Local or on the Buffer Memory, the physical address is reconstituted from the different fields of Buffer or List Pointer, used as operand. For performance reason, this operation is realized by a specialized hardware component called Memory Address Generator (MAG,139).

Direct Memory Access Controller (DMA, 134))

The use of a Direct Memory Access Controller (DMA, 134) jointly with a processor is well-known in the state of the art. Its role is to quickly move the data packets between the IO devices (137, 138) and the Buffer Memory (132) without the processor (SPP) intervention. The DMA module consists of two independent programmable channels. The IO devices present their service requests (SR1, SR2) to the DMA which controls the access to the Buffer Memory (132). The processor intervention is needed only at buffer and packet boundaries. The data streams between the two IO devices and the Buffer Memory is processed in parallel with the code. Up two IO operations can be multiplexed on the BMIO bus; one with IO1 and the other with IO2. For that the DMA manages the two DMA Pointers (D_PTR1 and D_PTR2) which are nothing else than Buffer Pointers.

Input Output Subsystems

The Specific Purpose Processor (SPP) is considered as the "master" and it establishes the connections.

The IO devices and the Buffer Memory are controlled either directly by the processor code, or via the DMA.

The code intervention can be forced by the IO devices via the Interrupt mechanism in case of buffer or packet boundary.

Data Reception and Transmission

Various processing can be made on Buffer and List Pointers:

Incrementing a Buffer Pointer,

Closing a buffer,

Accessing a List Prefix,

Attaching an element to a list,

Detaching an element from a list.

Operations on Pointers

Some operations on pointers are performed by the processor code, others by the Direct Memory Access (DMA).

The writing and reading of the Buffer Pointers is exclusively the fact of the DMA (134).

The writing in the Buffer Memory: At the reception of a service request (SR2) from an IO2, the DMA has access to the Buffer Memory (132) by means of the address contained in the Pointer 2 (D_PTR2, 136). The DMA Pointer 2 is provided by the processor (SPP) and is nothing else than a Buffer Pointer (B_PTR). The DMA orders simultaneously the IO2 (138) to present a data element on the BMIO bus and the Buffer Memory (132) to write this data element in the buffer identified by the DMA Pointer. Data are filled into the buffer starting an address chosen by the code and ending at the bottom of the buffer except the last buffer (of a packet) where data may end at any place. When the buffer is full, the DMA demands from the processor a new Buffer Pointer through an interrupt mechanism (IO2_EOB routine). A similar procedure is used when the IO2 detects the end of a packet (IO2_EOP routine)).

The Reading in the Buffer Memory: At the reception of a service request (SRI) from an IO1, the DMA addresses the Buffer Memory (132) by means of the DMA Pointer 1 (D_PTRI, 136). The DMA Pointer 1 is provided by the processor (SPP). The DMA orders simultaneously the Buffer Memory (132) to present on the BMIO bus a data element in the buffer identified by the DMA Pointer and the IO1 device (138) to read this data element. When the buffer is empty, the DMA demands a new Buffer Pointer from the processor through an interrupt mechanism (IO1_EOB routine). A similar procedure is used when the DMA detects a end of packet (IO1_EOP routine). After the data transfer, the Buffer Pointer is released in the Free Buffer List, Packet List and Queue List are updated accordingly.

Packet and Queue Pointers are managed by the processor code:

the Code intervention is forced by the IO devices and DMA via the Interrupt mechanism in case of buffer or packet boundary.

the buffer and packet queueing and dequeueing mechanisms are executed under the control of the processor in the Local Memory.

Interrupts

The Interrupt mechanism is the way real time is supported by the Specific Purpose Processor (SPP). An Interrupt is a break, due to particular events, in the normal sequence of the processor code. The events which can cause this Interrupt are the service requests from the IO devices associated with specific conditions such as end of buffer, end of packet . . . . At each specific Interrupt corresponds a specific routine which cannot be interrupted by another routine. It is very important to have interrupt routines as short as possible to avoid overrun/underrun problems.

1. The following Interrupts are used by the Line and Switch Transmitters (IO1).

IO1_EOB:

Condition: when serving an output IO1, the DMA has emptied a buffer (which is not the last buffer of the packet/segment—the Buffer Pointer is not flagged EOS or EOP) the

DMA Pointer (D_PTR1) raises a IO1_EOB interrupt which triggers a IO1_EOB routine.

Routine: this routine

releases the pointer of the just emptied buffer in the Free Buffer List. A new pointer is dequeued from the Output Buffer List (OB_LIST) and passed to the DMA Pointer (D_PTR1).

IO1_EOP:

Condition: when serving an output IO1, the DMA has emptied a packet (the last buffer of a packet which pointer contains the EOP flag ON), the DMA Pointer (D_PTR1) raises a IO1_EOP interrupt which triggers a IO1_EOP routine.

Routine: the routine:

releases the current and last Buffer Pointer of the Output Buffer List (OB_LIST) in the Free Buffer List (FBL).

releases the current Packet Pointer in the Free Packet List (FPL)

detaches the current Output Packet Pointer (OP_PTR) from the Output Packet List (OP_LIST).

dequeues the next packet and its pointer from the Output Packet List

2. These Interrupts are used by the Line and the Switch Receiver (IO2).

$IO2_{13}$ EOB:

Condition: when serving an input IO2, the DMA has detected a Buffer Full condition (That buffer is not the last buffer of a packet), the DMA Pointer (D_PTR2) raises a IO2_EOB interrupt which triggers a IO2_EOB routine.

Routine this routine:

stores the pointer of the just filled up buffer in a pre-allocated Input Buffer List (IB_LIST) where all buffers of the same packet are stacked.

updates the Input Buffer List Prefix area.

provides the DMA Pointer (D_PTR2) with a new Buffer Pointer for continuing the reception of the data of the same packet. Free Buffer Pointers are managed by the Free Buffer List (FBL).

when for the same packet, the Buffer List is full, the segmentation takes place.

IO2_EOP:

Condition: when serving an input IO2, the DMA has completed the reception of a packet (the last buffer of the packet which pointer is flagged EOP by the hardware), the DMA Pointer raises a IO2_EOP interrupt which triggers a IO2_EOP routine. The code intervention is required to provide a new Buffer Pointer for the reception of a new packet.

Routine: this routine:

stores the pointer of the last filled buffer in a pre-allocated Input Buffer List (IB_LIST) where all buffers of the same packet are automatically stacked. The last Buffer Pointer of a packet is flagged EOP by the DMA Pointer (D_PTR2).

updates the Input Buffer List Prefix area.

the Packet Pointer of the Input Buffer List is queued in the Input Packet List (IP_LIST).

provides the DMA Pointer (D_PTR2) with a new Buffer Pointer (B_PTR) for the reception of the next packet.

Point-to-Point and Multi-Point Routing

Multicasting means a packet should be sent to at least two destinations:

to different Transmit Adapters (via the Switch) in a Receive Adapter and when needed to the General Purpose Processor (GPP).

to output Trunks or Ports in a Transmit Adapter and when needed to the General Purpose Processor (GPP).

The general principles of the point-to-point and multi-points routing according to the present invention can be summarized as follows:

Data packets are never copied, only Packet Pointers are copied for each destination:

Space in Buffer Memory is saved.

The number of instructions required for the background (delayed) process is significantly reduced improving the packet throughput (number of packets per seconds that the adapter is able to transmit).

The routing is independent of the packets length.

No overhead generated by the multicasting mechanism in the interrupt (real time) procedures.

The underrun/overrun problems on the output device IO1 are reduced.

The efficiency of the adapter in term data throughput (bits per second) is significantly improved.

Each output is processed independently by means of interrupt (real time) routines:

Lines are managed in real time.

Lines of different speed or protocol can be supported in parallel.

The release of the resources is entirely realized on a no priority mode in the background (delayed) process.

Output Queues

Each destination has its own Output Queue where the packets waiting for transmission are stacked. The number of queues is limited to the number of destinations:

In the Receive Adapter (401), there is one Output Queue per adapter on the Switch (403) plus one specific Output Queue dedicated to the General Purpose Processor (GPP, 409).

Likewise in the Transmit Adapter (402), there is one Output Queue per Trunk or line in output of the Line Interface (416) plus one specific Output Queue dedicated to the General Purpose Processor (GPP, 409).

The Output Queues are located in the Local Memory (131). They are managed by the Specific Purpose Processor (SPP) by means of the background and interrupt (EOB, EOP) procedures detailed in flow charts of FIGS. 15, 16 and 17.

Output Queue Table

As shown in FIG. 10, during the transmission of the data packets, the state of each Output Queue is stored in a specific table called Output Queue Table (100). Each Output Queue is represented by one record comprising:

the Queue Pointer (104)(Q_PTR) identifying the queue.

the Current Packet Pointer (105) (cur P_PTR) identifying, when the Output Queue is ACTIVE, the packet to transmit.

the Current Buffer Pointer (106) (cur B_PTR) identifying, when the Output Queue is ACTIVE, the buffer to transmit.

the state (107) (ST) of the Output Queue:

ACTIVE state: a START TRANSMISSION has been sent to the IO1. The packet identified by the Current Packet Pointer (cur P_PTR) is transmitted from the Buffer Memory to the selected output under the control of the DMA.

NON ACTIVE state: the transmission of the previous packet is terminated and the Output Queue is empty.

Packet Processing

FIG. 14 gives a general view of the packet processing mechanism in a programmable high speed adapter. The receive, routing and transmission process involves the steps of:

(140) Reassembling each packet received in an Input Buffer List and initializing the Buffer List Prefix with the routing information. Packets in error are discarded.

(141) Creating for each packet a specific Buffer List identified by a Packet Pointer (IP1 . . . Ipm).

(142) Processing the packets according to their routing mode with a local access to a Routing Table (148).

(143) Queueing the Packet Pointers in the Output Queues (OQ1 . . . OQN) corresponding to the destination of the packets (145). Packets intended for the GPP (146) are stacked in a specific Output Queue (144)

(149) Triggering an interrupt routine each time a new buffer is requested. Interrupts are triggered asynchronously by the DMA on the occurrence of specific events: End Of Buffer (EOB), End Of Packet (EOP).

(1410) Transmitting the data from the Buffer Memory (BM,147) to the output device (IO1) under the control of the DMA.

(1411) Scanning the outputs (145).

Background Procedure

FIG. 15 is a flow chart illustrating the the packet routing and multicasting procedure represented in FIG. 14 (142):

(150) The processor waits for the next reassembled packet.

(151) As soon a packet is ready to be processed, its Packet Pointer (P_PTR) is retrieved. The Buffer List Prefix identified by said Packet Pointer (P_PTR) is updated. FIG. 10 details the different elements of the routing mechanism:

(102) The determination of the different destinations is given through a Routing Table (156) accessed by the Routing Field (802) of the packet header. In case of multi-points routing, Multicast Trees are predefined in each nodes at the connection set-up. Routing Tables (156) are maintained dynamically. That means that when a new connection is established or an old one is terminated the tables are updated (the database of network topology can of course be maintained quite separately). The Multicast Counter (MCC) in the Buffer List Prefix is set to its initial value according to the number of destinations given in the Routing Table (156). This value is equal to one for a point-to-point routing and superior to one for a multi-points routing mode.

(101) The Packet Pointer (P_PTR) (with the original HEAD and TAIL values) is saved in the Buffer List Prefix for a further release of the Buffers in the Buffer Memory.

(152) The Packet Pointer (P_PTR) corresponding to the packet to route or multicast is queued (the TAIL value of the Queue Pointer is incremented) in the first selected Output Queue (103).

(153) If the Output Queue is in an ACTIVE state, the process start again with with the next selected Output Queue (if any).

(154) If the Output Queue is in an NON ACTIVE state (ST), the first Packet Pointer of the Output Queue is dequeued (the HEAD value of the Queue Pointer is incremented) and saved as Current Packet Pointer (cur P_PTR) in the Output Queue Table. Likewise, the first Buffer Pointer (B_PTR) of the corresponding Buffer List is dequeued (the HEAD value of the Packet Pointer is incremented) and saved as Current Buffer Pointer (cur B_PTR) in this same Output Queue Table. Then, the Specific Purpose Processor (SPP) sends to the IO1 a request to start the transmission of the Current Packet (START TRANSMISSION). The Output Queue is set in an ACTIVE state (ST) in the Output Queue Table. The process goes on with the next selected Output Queue (if any).

(155) When all the selected outputs have been successively processed the packet routing and multicasting procedure releases all resources attached to the packets and buffers already transmitted (157). It is important to note that this release takes place in the background procedure without overloading the interrupt procedures.

(150) Once the release terminated the procedure returns to its initial waiting state.

Interrupt Procedures

1. Packet Level (IO1_EOP)

FIG. 16 is a flow chart illustrating the interrupt procedure at the packet level as represented in FIG. 14 (149). On the occurrence of an End Of Packet interrupt (EOP):

(160) The Current Packet Pointer (cur P_PTR) corresponding to the last Output Queue processed (OQn) is extracted from the Output Queue Table.

(161) The Current Packet Pointer is queued in a Release Queue for a further release of the Packet Pointer in the Free Packet List and of the Buffer Pointers in the Free Buffer List. This release process is entirely realized in the background procedure (157) to reduce to a minimum the number of instructions in the interrupt procedures.

(162) The contents of the Output Queue (OQn) is tested:

(165) If Output Queue (OQn) is empty, then it is set in a NON ACTIVE state (ST) in the Output Queue Table.

(163) If Output Queue (OQn) is not empty, then the next Packet Pointer is dequeued (the HEAD value in the Queue Pointer is incremented) (166). The first Buffer Pointer of the Buffer List is dequeued (the HEAD value of the Packet Pointer is incremented) and saved (Current Buffer Pointer) in the Output Queue Table with the Packet Pointer (Current Packet Pointer). Once the Current Buffer Pointer (cur B_PTR) and the Current Packet Pointer (cur P_PTR) are identified, the Specific Purpose Processor (SPP) sends to the IO1 a request to start the transmission of the Current Packet (START TRANSMISSION).

2. Buffer Level (IO1_EOB)

FIG. 17 is a flow chart illustrating the interrupt procedure at the buffer level as represented in FIG. 14 (149). On the occurrence of an End Of Buffer interrupt (EOB):

(170) The Current Packet Pointer (cur P_PTR) corresponding to the last Output Queue processed (OQn) is extracted from the Output Queue Table.

(171) The next Buffer Pointer (B_PTR) in the Buffer List is dequeued (the HEAD value of the Packet Pointer is incremented) This Buffer Pointer is first, set into the DMA Pointer (D_PTR1) attached to the IO1 device and second, saved as Current Buffer Pointer (cur B_PTR) in the Output Queue Table. The Packet Pointer is also saved in the Output Queue Table as Current Packet Pointer.

Release of Resources

FIG. 19 is a flow chart illustrating the release of the resources attached to the packets after their transmission as represented in FIG. 15 (157). This process is entirely located in the background procedure to minimize the number of instructions during the interrupts.

(190) The contents of the Release Queue located in the Local Memory (131) is tested:

If the Release Queue is empty, the release process is terminated and the background procedure can return in its initial waiting state (150).

If the Release Queue is not empty then:

(191) A Packet Pointer is dequeued from the Release Queue (the HEAD value of the Release Queue Pointer is incremented).

(191) The Multicast Counter (MCC) is retrieved from the Buffer List Prefix.

(191) The Multicast Counter (MCC) is decremented by one and tested (192):

(193) If the Multicast Counter (MCC) value is equal to zero, then the transmission of the packet is terminated. All the resources attached to this packet must be released. The original Packet Pointer (P_PTR) (with the initial HEAD and TAIL values) previously saved is retrieved from the Buffer List Prefix. The Packet Pointer and Buffers Pointers are released respectively from the Free Packet List (FPL) and from the Free Buffer List (FBL).

If the Multicast Counter (MCC) value is not equal to zero, then the multicast process is not terminated.

(190) The contents of the Release Queue is tested one more time:

If the Release Queue is not empty, the procedure goes on with a new Packet Pointer (191).

If the Release Queue is empty, the release process is terminated and the background procedure can return in its initial waiting state (150).

Data Transmission

FIG. 18 illustrates the data flow between the Specific Purpose Processor (SPP), the Direct Access Memory (DMA) and IO1 device. First, the background procedure already described in FIG. 15 is in a waiting state. As soon as a packet is ready to be transmitted, the Packet Pointer is copied in the appropriate Output Queue n. This Output Queue n is set in an ACTIVE state and the SPP informs the IO1 device that it is ready to start the transmission on the output n (START TRANSMISSION n). At the reception of this message, the IO1 responds to the DMA with a Service Request (SR1 n) to request a first Buffer Pointer. the DMA triggers an End Of Buffer interrupt (IO1_EOB n as shown in FIG. 17) to extract from the Output Queue Table the Current Buffer Pointer (cur B_PTR n) of the Output Queue n. The DMA orders simultaneously the Buffer Memory (BM) to present the buffer identified by said Current Buffer Pointer on the BMIO and the IO1 to read this buffer (DS1 n Data Service message). The process goes on with the next buffer until the end of the packet. At this time the DMA raises an End Of Packet interrupt (IO1_EOP n as shown in FIG. 16) and informs the IO1 device (EOP1 n).

It is important to notice that all Output Queues are processed independently and in parallel under control of the IO1 device according to the different protocols and speeds used on the output lines.

We claim:

1. A line adapter for a packet switching node in a communication network, including a programmable processing means (SPP) for receiving and transmiting data packets of fixed or variable length to one or more outputs, said line adapters comprising:

a first storing means for buffering each data packet in one or more buffers;

means for identifying said buffers,

a second storing means including means for queuing said buffer identifiers in buffer lists, each buffer list identifying a data packet;

means for identifying said buffer lists;

means for queuing, in said second storing means, buffer list identifiers in packet lists, each packet list identifying a queue;

means for identifying said packet lists;

means for processing a routing header of each data packet;

means for associating with each output an output queue for stacking the packet list identifiers of the data packets to transmit on said outputs;

means for routing the data packets to one or a plurality of outputs;

means for processing said routing header further comprising means for determining and selecting the output queue corresponding to the destination of each data packet;

said means for routing further comprising:

means for copying the buffer list identifier of the data packet to transmit in the output queue corresponding to said selected output;

means for handling each data request for said outputs in real time;

means for releasing said buffers in said first storing means, and said buffer identifiers and said buffer list identifiers from said second storing means;

said means for handling data requests further comprising means for handling independently the output queues, which further includes:

means for dequeuing in parallel the packet identifiers requested by the outputs; and

means for dequeuing the buffer identifiers stacked in the buffer list identified by said packet identifiers.

2. The line adapter according to claim 1 wherein:

said first storing means includes means for writing and reading said data packets in buffers of fixed length independently of said programmable processing means; and

said second storing means includes means for separately storing said buffer lists and said packet lists under control of said programmable processing means.

3. The line adapter according to claim 1 wherein said programmable processing means (SPP) comprises:

a background procedure for processing the data packets,

a plurality of real time procedures triggered under control of each output to request the contents of a new buffer, said real time procedures being independent of the routing mode used.

4. The line adapter according to claim 3 wherein said means for releasing in said means for routing is executed in said background procedure.

5. The line adapter according to claim 1 wherein:

said means for identifying said buffers includes buffer pointers (B_PTR) identifying said buffers and stacked in one or more buffer lists (B_LIST);

said means for identifying said buffer lists includes packet pointers (P_PTR) identifying said buffer lists (B_LIST) and stacked in one or more packet lists (P_LIST); and

said means for identifying said packet lists include queue pointers (Q_PTR) identifying said packet lists (P_LIST) and stacked in one or more queue lists (Q_LIST); and

wherein each buffer list, packet list and queue list includes a prefix for storing information related to the data contained in said each buffer list, packet list and queue list.

6. The line adapter according to claim 5 wherein said buffer list prefix includes control and routing information contained in the data packet header.

7. The line adapter according to claim 6 wherein said buffer list prefix further includes a multicast counter (MCC COUNTER) and a packet pointer (P_PTR).

8. The line adapter according to claim 7 wherein a status of each output queue is stored in an output queue table located in said second storing means, said status including:

the output queue pointer (Q_PTR),

a current packet pointer (cur P_PTR) for the last packet dequeued from the output queue;

a current buffer pointer (cur B_PTR) for the last buffer dequeued from the current packet pointer; and

a state of the output queue depending on if the output queue is active, an output queue being active during the time said data packet identified by said current packet pointer is transmitted to said selected output.

9. The line adapter according to claim 8 wherein said background procedure includes:

means for detecting a new buffer list;

means for reading the routing information in the buffer list prefix;

means for initializing the multicast counter in the buffer list prefix with the number of outputs towards which the data packet must be transmitted;

means for copying the packet pointer of the buffer list in its own prefix;

means for selecting an output queue according to the destination of the data packet, queuing in said output queue the packet pointer and testing the state of said output queue, wherein:

if the state is active a new output queue is selected according to the other destinations of the data packet;

if the state is not active:

a packet pointer is dequeued from said output queue and saved as the current packet pointer in said output queue table;

a buffer pointer is dequeued from the buffer list identified by said current packet pointer and saved as the current buffer pointer in said output queue table;

a message is sent to the output;

the output queue is set in an active state; and

a new output queue is selected according to the other destinations of the data packet;

when said packet pointer has been queued in all selected output queues the background process returns to its waiting state.

10. The line adapter according to claim 9 wherein said plurality of real time procedures comprise:

a first real time procedure triggered by the outputs on request for a new buffer including:

means for getting the packet pointer corresponding to the last output queue processed in the output queue table;

means for dequeing the next buffer pointer in the buffer list identified by said packet pointer;

means for saving said packet pointer into the output queue table as the current packet pointer;

means for saving said buffer pointer into the output queue table as the current buffer pointer;

a second real time procedure triggered by the outputs on request for a new packet including:

means for getting the packet pointer corresponding to the last output queue processed in the output queue table;

means for queuing said packet pointer in a release queue for a delayed release;

means for testing if the output queue is empty or not, wherein:

if said output queue is empty, said output queue is set in a non-active state;

if said output queue is not empty:

a next packet pointer is dequeued from said output queue;

said next packet pointer is saved as the current packet pointer in the output queue table;

a first buffer pointer of the buffer list identified by said packet pointer is saved as the current buffer pointer in the output queue table; and

a message is sent to the output associated with said output queue.

11. The line adapter according to claim 1 wherein the management of the buffers in said first storing means is realized by means of a first permanent list (free buffer list) containing all of the buffer pointers.

12. The line adapter according to claim 1 wherein the management of the buffer lists in said second storing means is realized by means of a second permanent list (free packet list) containing all of the packet pointers.

13. The line adapter according to claim 1 wherein said releasing means in said background procedure comprises:

means for testing the contents of the release queue:

if the release queue is empty, then terminating the release;

if the release queue is not empty:

dequeuing a packet pointer from the release queue;

decrementing the buffer list prefix counter by one;

testing if the multicast counter value is equal to zero,

if the multicast counter value is not zero, then testing the contents of the release queue:

if the release queue is empty, then terminating the release;

if the release queue is not empty, then dequeuing the next packet pointer from the release queue;

if the multicast counter value is zero:

retrieving the original packet pointer from the buffer list prefix identified by said packet pointer;

releasing the buffer list and buffers identified by said packet pointer from the free packet list and the free buffer list;

testing the contents of the release queue.

14. The line adapter according to claim 5 wherein each list pointer (packet pointer (P_PTR) or queue pointer (Q_PTR)) comprises:

a first field (LID) for identifying the list;

a second field for identifying a next pointer (TAIL) to stack in said list;

a third field for identifying a first pointer (HEAD) stacked in said list.

15. The line adapter according to claim 14 wherein said means for queuing comprises:

means for incrementing the TAIL field of the list pointer;

means for simultaneously storing the pointer identified by the TAIL field in the list identified by the LID field;

means for generating a list full indicator.

16. The line adapter according to claim 14 wherein said means for dequeuing comprises:

means for incrementing the HEAD field of the list pointer;

means for simultaneously reading the pointer identified by the HEAD field, in the list identified by the LID field;

means for generating a list empty indicator.

17. The line adapter according to claim 15 wherein said means for queuing further includes a means for testing said list full indicator, and said means for dequeuing further includes a means for testing said list empty indicator.

18. The line adapter according to claim 1 wherein said programmable processing means includes:

an arithmetical and logical unit,

a register file,

a sequencer,

an instruction file,

a direct memory access controller module, and

a physical memory address generator.

19. A routing method in a line adapter for a packet switching node in a communication network, including programmable processing means (SPP) for receiving and transmitting data packets of fixed or variable length to one or more outputs, said routing method comprising the steps of:

buffering each data packet in one or more buffers in a first storing means;

identifying said buffers;

queuing, in a second storing means, said buffer identifiers in buffer lists, each buffer list identifying a data packet;

identifying said buffer lists;

queuing buffer list identifiers in packet lists in said second storing means, each packet list identifying a queue;

identifying said packet lists;

processing a routing header of each data packet;

associating with each output an output queue for stacking the packet list identifiers of the data packets to transmit on said outputs;

routing the data packets to one or a plurality of outputs;

said processing the routing header step further comprising the step of determining and selecting the output queue corresponding to the destination of each data packet;

said routing step further comprising the steps of:

copying the buffer list identifier of the data packet to transmit in the output queue corresponding to said selected output;

handling each data request for said outputs in real time;

releasing said buffers in said first storing means, and said buffer identifiers and buffer list identifiers from said second storing means;

said handling each data request step including the step of handling independently the output queues by

dequeuing in parallel the packet identifiers requested by the outputs; and

dequeuing the buffer identifiers stacked in the buffer list identified by said packet identifiers.

* * * * *